THE UNIVERSITY OF ALBERTA

A HIGH ACCURACY DIGITAL SAMPLING WATTMETER FOR DISTORTED AND VARYING FREQUENCY SIGNALS

by

DAVID JECTY NYARKO

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA

FALL 1989

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR:	DAVID JECTY NYARKO
TITLE OF THESIS:	A HIGH ACCURACY DIGITAL SAMPLING WATTMETER FOR DISTORTED AND VARYING FREQUENCY SIGNALS
DEGREE:	MASTER OF SCIENCE
YEAR THIS DEGREE GRANTED:	FALL, 1989

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

.....

(Student's signature)

Student's permanent address:

2B 8916 - 112 Street

EDMONTON, ALBERTA

T6G 2C5

Date: 31st August 1989

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled A HIGH ACCURACY DIGITAL SAMPLING WATTMETER FOR DISTORTED AND VARYING FREQUENCY SIGNALS submitted by DAVID JECTY NYARKO in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE IN ELECTRICAL ENGINEERING.

.....

(Supervisor)

.....

.....

Date: .21..Aug..1989....

ABSTRACT

This thesis describes an approach to the implementation of digital sampling wattmeters. A numerical integration algorithm employing a trapezoidal rule with an end correction, is shown to considerably improve the accuracy of such instruments. This results in a more precise measure ent of distorted and varying frequency waveforms. This approach is employed in the design of a digital sampling wattmeter design primarily intended for power system measurements.

The stand-alone system based on an 8Mhz 68000 microprocessor system, is capable of measuring distorted power signals with a fundamental frequency from dc to 1khz. Auxiliary functions in this GPIB compatible and keypad programmable instrument include voltage, current , power factor and frequency measurements in various modes.

ACKNOWLEDGEMENT

I would like to thank my sponsors, the Association of Universities and Colleges of Canada, for their financial support for my MSc. program. I would also like to thank my supervisor, Dr. Keith Stromsmoe, for his help and encouragement in preparing this thesis.

TABLE OF CONTENTS

ABSTRACT iv

ACKNOWLEDGEMENT v

LIST OF FIGURES x

LIST OF SYMBOLS xiv

1. INTRODUCTION 1

2. THEORY 3

2.1 Analysis of sampled waveforms 5

2.2 Error Determination 6

2.3 Error expressions for various processing methods 8

2.3.1 Average of samples 10

2.3.2 Trapezoidal Rule 12

2.3.3 NBS approach 14

2.3.4 Modified Trapezoidal Approach 15

2.4 Determination of delta 17

2.5 Computation of the desired parameters . . . 18

2.6 Summary of results 20

3. **DIGITAL SAMPLING WATTMETER HARDWARE** 35

3.1 CPU module 35

3.1.1 Microprocessor circuitry 36

3.1.2 Memory circuitry 37

3.1.3 Interrupt and Watchdog timer circuitry 37

3.2 I/O module 38

3.3 Timing Controller module 40

3.4 Analog-to-Digital interface module 42

3.5 Input-conditioning circuitry 43

3.5.1 Phase 1 circuitry 43

3.5.2 Phase 2 circuitry 44

3.5.3 Analog switches circuitry 44

3.5.4 Trigger circuitry 45

4. **DIGITAL SAMPLING WATTMETER SOFTWARE** 59

4.1 Software Overview 59

4.2 Initialization and Memory test 59

4.3 The main program 61

4.3.2 Command preprocessor 61

4.3.3 Module 0 63

4.3.4 Module 1 63

4.3.5 Module 2 63

4.3.6 Module 3 64

4.3.7 Module 4 64

4.3.8 Module 5 65
4.3.9 Module 6 66
4.3.10 Module 7 66
4.3.11 Module 8 67
4.3.12 Module 9 67
4.3.13 Module 10 68
4.3.14 Module 11 69
4.3.15 Module 12 70
4.3.16 Module 13 71
4.3.17 Module 14 71
4.3.18 Module 15 72
4.3.19 Module 16 72
4.3.20 Module 17 73
4.3.21 Module 18 73
4.3.22 Module 19 74
4.3.23 Module 20 74
4.3.24 Module 21 75
4.3.25 Module 22 75
4.3.26 Module 23 75
4.3.27 Module 24 76
4.3.28 Module 25 76
4.3.29 Module 26 77
4.3.30 Module 27 77
4.3.31 Module 28 77
4.3.32 Module 29 78

5. SYSTEM ROUTINES 108

5.1 Acquisition routines 108

5.2 Computational routines 110

5.3 Utility routines 111

6. SYSTEM OPERATION, PERFORMANCE AND SPECIFICATIONS 120

6.1 Calibration 120

6.2 Scale factor calculations 122

6.3 Performance checks 123

6.4 Sources of error 128

6.5 Specifications 128

6.6 Suggestions for improvements 129

7. CONCLUSION 130

REFERENCES 131

APPENDIX A WATTMETER CIRCUIT DIAGRAMS 133

APPENDIX B SOFTWARE LISTING 158

LIST OF FIGURES

Figure	Description	Page
2.1	Graph of alpha ratios vs delta (average/procedure used)	22
2.2	Graph of alpha ratios vs delta (trapezoidal/procedure used)	23
2.3	Graph of alpha ratios vs delta (NBS/procedure used)	24
2.4	Graph of beta ratios vs delta (average/procedure used)	25
2.5	Graph of beta ratios vs delta (trapezoidal/procedure used)	26
2.6	Graph of beta ratios vs delta (NBS/procedure used)	27
2.7	Graph of alpha vs delta for the procedure used	28
2.8	Graph of beta vs delta for the procedure used	29
2.9	Graph of absolute value of error coefficients vs harmonic number for a frequency of 59.925 hz	30
2.10	Graph of absolute value of error coefficients vs harmonic number for a frequency of 59.98 hz	31
2.11	Graph of absolute value of error coefficients	

	vs harmonic number for a frequency of 60.035 hz	32
2.12	Graph of absolute value of the ratio alpha/beta vs harmonic number	33
3.1	Block diagram of wattmeter	46
3.2	Block diagram of CPU module	46
3.3	Block diagram of microprocessor circuitry	47
3.4	System memory map	48
3.5	Detailed memory map of I/O peripherals and A/Board	49
3.6	Block diagram of memory circuitry	50
3.7	Block diagram of interrupt and watchdog timer circuitry	51
3.8	Block diagram of I/O module	52
3.9	Block diagram of the timing controller module	53
3.10	A/D Interface timing diagram	54
3.11	Block diagram of the Analog-to-Digital interface module	55
3.12	Block diagram of the input conditioning module	56
3.13	Block diagram of the phase 1 circuitry	56
3.14	Block diagram of the phase 2 circuitry	57
3.15	Block diagram of the analog switches circuitry	57

3.16	Block diagram of the trigger circuitry . . .	58
4.1	Flowchart of the system software	79
4.2	Flowchart of the initialization and memory test	80
4.3	Flowchart of the main program	81
4.4	Detailed main menu displays	82
4.5	Flowchart of module 0	83
4.6	Flowchart of module 1	84
4.7	Flowchart of module 2	85
4.8	Flowchart of module 3	86
4.9	Flowchart of module 4	87
4.10	Flowchart of module 5	88
4.11	Channel selection sub-menu	89
4.12	Flowchart of module 6	89
4.13	Flowchart of module 7	90
4.14	Flowchart of module 8	91
4.15	Flowchart of module 9	92
4.16	Flowchart of module 10	93
4.17	Flowchart of module 11	94
4.18	Flowchart of module 12	95
4.19	Flowchart of module 13	96
4.20	Flowchart of module 14	97
4.21	Flowchart of module 15	98
4.22	Flowchart of module 16	99
4.23	Flowchart of module 17	100

4.24 Flowchart of module 18 101

4.25 Flowchart of module 19 102

4.26 Flowchart of module 20 102

4.27 Flowchart of module 21 103

4.28 Flowchart of module 22 103

4.29 Flowchart of module 23 104

4.30 Flowchart of module 24 105

4.31 Flowchart of module 25 105

4.32 Flowchart of module 26 106

4.33 Flowchart of module 27 106

4.34 Flowchart of module 28 107

4.35 Flowchart of module 29 107

5.1 Crossing detector timing diagram 1 117

5.2 Crossing detector timing diagram 2 117

5.3 Flowchart of the input routine 118

5.4 Flowchart of the LCD display routine 119

6.1 Graph of rms value vs signal frequency (60 hz) 124

6.2 Graph of rms value vs frequency 125

6.3 Graph of wattmeter reading vs calibration meter reading (60 Watt Incandescent lamp) . . . 126

6.4 Graph of wattmeter reading vs calibration meter reading (Fluorescent lamp) 127

LIST OF SYMBOLS

Symbol	Description
α_k	The kth harmonic sine term error coefficient
β_k	The kth harmonic cosine term error coefficient
f	Frequency
i	Current
j	Sample number
k	Harmonic number
I_{rms}	Root-mean-square current
h	Sampling interval (angular measurement in radians)
P_{ac}	Active electrical power (AC)
P_{dc}	Electrical power (DC)
t	Time
T	Period
v	Voltage
V_{rms}	Root-mean-square voltage

1. INTRODUCTION

The widespread development and use of electric power systems with highly distorted waveforms has rendered the use of the relatively commonplace instruments, particularly the analog type, inappropriate. Basically, two classes of instruments exist [5], based on the computational method employed; the analog and the digital types. Hybrid instruments employing both techniques are also in existence.

Analog instruments although relatively inexpensive with regard to their digital counterparts, suffer from numerous deficiencies. The analog computation approach suffers from amplifier gain bandwidth limitations, matching of analog components as well as a low dynamic range and sensitivity. Furthermore, calibration is not easily carried out.

The alternative digital approach, used in digital sampling meters, entails sampling the input voltage and current signals and employing fully digital arithmetic procedures to compute the required parameters. This results in very precise results. It is also easily effected in digital stored-program systems such as microprocessor-based implementations.

Two distinct approaches to wattmeter sampling timing exist. In the synchronous approach, the sample timing and the waveform frequency are synchronised so that a block of samples represents an integral number of periods.

Alternatively, an asynchronous technique is employed, in which sampling is at a fixed rate independent of the waveform frequency. The first-mentioned approach has been widely employed and analyzed [1],[10],[13]. This approach, although easy to implement, results in accurate meters only if the signal frequency under consideration is stable. The computation of electrical power, voltage and current on systems operated from the electrical mains require an asynchronus technique for an adequate degree of accuracy. This approach is employed in the National Bureau of Standards calibration wattmeters [12]. Since the blocks of samples processed in this approach do not generally represent an integer number of periods, an allowance must be made for the ends of the block. The approach utilized, the modified trapezoidal approach, unlike other processing methods employed, uses all the samples obtained, in computing the required integral.

The device was developed to meet the following objectives.

1) Peak input voltage, current and maximum power of 350v, 10A and 3500W respectively.
2) Input signal frequency from dc to 1khz.
3) Frequency measurement range; 12 - 1000hz.
4) A user-friendly software interface.
5) An easy to maintain hardware.

2 THEORY

The determination of electrical power, voltage and current entails the computation of appropriately scaled integrals. Digital sampling meters utilize values from a discrete set approximating continuous signal values. The application of numerical integration techniques to the sample set results in an approximation to the signal parameters desired.

The fundamental equations for calculating the electrical dc power, the dc voltage, the dc current, the electrical ac power,and the root-mean-square voltage and current are respectively;

$$P_{dc} = \frac{1}{A}\int_0^A v\ i\ dt \qquad (2.1)$$

$$V_{dc} = \frac{1}{B}\int_0^B v\ dt \qquad (2.2)$$

$$I_{dc} = \frac{1}{C}\int_0^C i\ dt \qquad (2.3)$$

$$P_{ac} = \frac{1}{T}\int_0^T v\ i\ dt \qquad (2.4)$$

$$V_{rms} = \sqrt{\frac{1}{T}\int_0^T v^2\ dt} \qquad (2.5)$$

$$I_{rms} = \sqrt{\frac{1}{T}\int_0^T i^2\,dt} \qquad (2.6)$$

where A,B,C are arbitrary time intervals,

T is the period

v is the instantaneous voltage

and i is the instantaneous current

Equations 2.1 to 2.6 involve parameters which are functions of the average of integrals. In general, these equations can be represented as;

$$x = F[I(y(t))] \qquad (2.7)$$

where x is the parameter required

y(t) is the corresponding variable

I is an averaging function dependent on the integral of y(t)

and the function F, represents any extra processing required to compute the parameter.

For example, relating the above equation to equation 2.5;

$$x(t) = V_{rms}$$

$$y(t) = v^2$$

$$I(y(t)) = \frac{1}{T}\int_0^T v^2\,dt$$

and F is the square-root function.

2.1 Analysis of sampled waveforms

A convenient approach to the analysis of the periodic signals encountered in the measurement of the system parameters is the Fourier series technique.

A Fourier series expansion for a periodic signal y(t) is:

$$y(t) = a_0 + \sum_{k=1}^{\infty} a_k \sin\left(\frac{2\pi kt}{T}\right) + \sum_{k=1}^{\infty} b_k \cos\left(\frac{2\pi kt}{T}\right) \quad (2.8)$$

where T = the fundamental period

t = time

Sampling the above signal at regularly spaced intervals of length h radians, produces sampled values y_j which can be expressed as:

$$y_j = a_0 + \sum_{k=1}^{\infty} a_k \sin\left(kjh\right) + \sum_{k=1}^{\infty} b_k \cos\left(kjh\right) \quad (2.9)$$

$j = 0,1,2,..,n$ for a signal sampled at n+1 points

The above representation of an ac signal can also be used to represent the power signal which exists as a result of the microprocessor computations on the sampled values of voltage and current. This is due to the simultaneous sampling of the sinusoidal voltage and current signals.

2.2 Error Determination

The basic principle involved in the determination of the root mean square values of the voltage and current as well as the average power, is the application of a linear averaging operator **A** to the product of sampled values v_j^2, i_j^2 and $v_j i_j$ respectively. This operator computes the mean-value of an integral. It is thus the discrete equivalent of the continuous data averaging function I in equation 2.7. The existence of numerous algorithms for computing the integral of a discrete data set, thus result in a variety of expressions for **A**. These expressions for **A** are referred to in this thesis, as the processing methods. The error in any processing method indicates the deviation of the parameters obtained using that method to that obtained by computing the mean value of the integral with continuous signals.

Two schemes for the error determination exist: time domain and frequency domain analysis. The former is suitable for synchronous sampling implementations but difficult to interpret for asynchronous sampling schemes. The latter approach [14] however covers both sampling schemes easily, and is easily interpreted. This is due to the fact that the error coefficients are related directly to the signal harmonics. This approach is used in the subsequent equations.

Let y represent the products of the instantaneous values of the relevant continuous signals, and y_j represent

the products of the sampled values. The error E resulting from the application of the linear averaging operator A on y_j is:

$$E = Ay_j - \frac{1}{T}\int_0^T y(t)\,dt \qquad (2.10)$$

$$= A\sum_{k=1}^{\infty} a_k \sin(kjh) + A\sum_{k=1}^{\infty} b_k \cos(kjh) \qquad (2.11)$$

since $Aa_0 = a_0$

$$= \sum_{k=1}^{\infty} a_k\, A\sin(kjh) + \sum_{k=1}^{\infty} b_k\, A\cos(kjh) \qquad (2.12)$$

$$= \sum_{k=1}^{\infty} a_k\alpha_k + \sum_{k=1}^{\infty} b_k\beta_k \qquad (2.13)$$

The error coefficients α_k (alpha) and β_k (beta) are thus dependent on the processing method. The equation further indicates the dependence of the error on the Fourier series coefficients and hence the start of the sampling process.

For n+1 regularly spaced samples spanning n intervals of duration h, the following expression is obtained

$$(n + \Delta)h = 2m\pi \qquad (2.14)$$

where n,m are integers

and Δ (delta) is a dimensionless quantity.

2.3 Error expressions for various processing methods

Two distinct cases should be analyzed in the determination of the error expressions.

a) $kh = 2\pi p$ where p is a positive integer

b) $kh \neq 2\pi p$ where p is a positive integer

a) $\underline{kh = 2\pi p}$

In this case, a general error expression can be obtained for all processing methods which satisfy the relation ; $Aa_0 = a_0$.

$$\alpha_k = A \sin(jkh)$$
$$= A \sin(j2\pi p)$$
$$= 0$$

$$\beta_k = A \cos(jkh)$$
$$= A \cos(j2\pi p)$$
$$= 1$$

It should be noted that the error expressions obtained for α_k and β_k above, are applicable to a synchronous sampling scheme provided the following conditions are satisfied.

a) n+1 samples covering m periods

where m and n are coprime.

b) $k = p\left(\frac{n}{m}\right)$.

For example if 521 samples are taken in a period of a waveform, only the 520th ,1040th etc. harmonics will contribute to the error in measurement of the required

parameter.

b) $kh \neq 2\pi p$

In this instance, the error is dependent on the processing scheme. Four existing processing schemes will be considered; the average of samples, the trapezoidal rule, the NBS approach and the processing method used in this thesis.

In the average of samples approach the integral is approximated by a series of rectangles of height equal to the sampled values. This approach does not take into account the last sampled value. No correction is made to account for cases in which the total sampling interval does not cover an integral number of cycles. It is however the most widely used approach. The trapezoidal rule approximates the integral using a series of trapezoids. All the samples are used in the computation of the integral. Like the average method, no correction is made for the sampling interval. The NBS method is simply an average method including an end correction Δ in the range [0-1]. Finally, the approach used in this thesis is simply a trapezoidal rule with an end correction. Unlike the NBS method, the end correction Δ is restricted to the range [-0.5,0.5] by adjusting the sample size accordingly. This method will be referred to as the modified trapezoidal approach.

2.3.1 Average of samples

The average value for n+1 samples [2],[6],[8] is given by;

$$Ay_j = \frac{1}{n}\sum_{j=0}^{n-1} y_j \qquad (2.15)$$

The corresponding error coefficient relating to the sine term of the error expression is:

$$\overset{A}{\alpha}_k = A\sin(jkh)$$

$$= A\left[\mathrm{Im}\left(e^{ijkh}\right)\right]$$

$$= A\left[\mathrm{Im}\left(e^{ij\theta}\right)\right]$$

where $\theta = kh$

$$= \frac{1}{n}\sum_{j=0}^{n-1}\mathrm{Im}\left(e^{ij\theta}\right)$$

$$= \frac{1}{n}\,\mathrm{Im}\left(\frac{1 - e^{in\theta}}{1 - e^{i\theta}}\right)$$

$$= \frac{1}{n}\,\mathrm{Im}\left[\left(\frac{1-\cos(n\theta)-i\sin(n\theta)}{1-\cos\theta-i\sin(n\theta)}\right)\left(\frac{1-\cos\theta+i\sin\theta}{1-\cos\theta+i\sin\theta}\right)\right]$$

$$= \frac{1}{n}\left[\frac{\sin\theta(1-\cos(n\theta)) - \sin(n\theta)(1-\cos\theta)}{(1-\cos\theta)^2 + \sin^2\theta}\right]$$

now $(n + \Delta)h = 2m\pi$

$\Rightarrow (n + \Delta)kh = 2mk\pi$

$nkh = 2mk\pi - \Delta kh$

$\Rightarrow \quad n\theta = 2mk\pi - \Delta kh$

$\Rightarrow \quad \sin(n\theta) = -\sin(\Delta kh)$

and $\cos(n\theta) = \cos(\Delta kh)$

hence;

$$\overset{A}{\alpha}_k = \frac{1}{n}\left[\frac{\sin\theta(1-\cos(\Delta\theta)) + \sin(\Delta\theta)(1-\cos\theta)}{2(1-\cos\theta)}\right]$$

$$= \frac{1}{n}\left[\cot\left(\frac{k\,h}{2}\right)\sin^2\left(\frac{\Delta kh}{2}\right) + \frac{\sin(\Delta kh)}{2}\right]$$

The corresponding error coefficient relating to the cosine term of the error expression is:

$$\overset{A}{\beta}_k = A\cos(jkh)$$

$$= A\left[\mathrm{Re}\left(e^{ijkh}\right)\right]$$

$$= A\left[\mathrm{Re}\left(e^{ij\theta}\right)\right] \qquad \text{where } \theta = kh$$

$$= \frac{1}{n}\sum_{j=0}^{n-1}\mathrm{Re}\left(e^{ij\theta}\right)$$

$$= \frac{1}{n}\,\mathrm{Re}\left(\frac{1 - e^{in\theta}}{1 - e^{i\theta}}\right)$$

$$= \frac{1}{n}\,\mathrm{Re}\left[\left(\frac{1-\cos(n\theta)-i\sin(n\theta)}{1-\cos\theta-i\sin(n\theta)}\right)\left(\frac{1-\cos\theta+i\sin\theta}{1-\cos\theta+i\sin\theta}\right)\right]$$

$$= \frac{1}{n}\left[\frac{(1-\cos(n\theta)(1-\cos\theta) + \sin\theta\sin(n\theta)}{(1 - \cos\theta)^2 + \sin^2\theta}\right]$$

now $(n + \Delta)h = 2m\pi$

$\Rightarrow (n + \Delta)kh = 2mk\pi$

$nkh = 2mk\pi - \Delta kh$

$\Rightarrow n\theta = 2mk\pi - \Delta kh$

$\Rightarrow \sin(n\theta) = -\sin(\Delta kh)$

and $\cos(n\theta) = \cos(\Delta kh)$

hence;

$$\beta_k^A = \frac{1}{n}\left[\frac{(1-\cos(\Delta\theta))(1-\cos\theta) - \sin\theta\sin(\Delta\theta)}{2(1 - \cos\theta)}\right]$$

$$= \frac{1}{n}\left[\sin^2\left(\frac{\Delta kh}{2}\right) - \frac{1}{2}\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh)\right]$$

2.3.2 Trapezoidal rule

The expression for n+1 samples using the trapezoidal rule is:

$$Ay_j = \frac{1}{n}\left(0.5y_0 + \sum_{j=1}^{n-1} y_j + 0.5y_n\right) \qquad (2.16)$$

$$= \frac{1}{n}\left(\sum_{j=0}^{n-1} y_j - 0.5y_0 + 0.5y_n\right)$$

The corresponding error coefficient relating to the sine term of the error expression is:

$$\Rightarrow \quad \alpha_k^T = \mathbf{A}\ \sin(jkh)$$

$$= \frac{1}{n}\left[\cot\left(\frac{k\,h}{2}\right)\ \sin^2\left(\frac{\Delta kh}{2}\right) + \frac{\sin(\Delta kh)}{2} - \frac{\sin 0}{2} + \frac{\sin(nkh)}{2}\right]$$

$$= \frac{1}{n}\left[\cot\left(\frac{k\,h}{2}\right)\ \sin^2\left(\frac{\Delta kh}{2}\right) + \frac{\sin(\Delta kh)}{2} - \frac{\sin 0}{2} - \frac{\sin(\Delta kh)}{2}\right]$$

$$= \frac{1}{n}\left[\cot\left(\frac{k\,h}{2}\right)\sin^2\left(\frac{\Delta kh}{2}\right)\right]$$

The corresponding error coefficient relating to the cosine term of the error expression is:

$$\beta_k^T = \mathbf{A}\ \cos(jkh)$$

$$= \frac{1}{n}\left[\sin^2\left(\frac{\Delta kh}{2}\right) - \frac{1}{2}\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh) - \frac{\cos 0}{2} + \frac{\cos(nkh)}{2}\right]$$

$$= \frac{1}{n}\left[\sin^2\left(\frac{\Delta kh}{2}\right) - \frac{1}{2}\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh) \right.$$

$$\left. - \frac{\cos 0}{2} + \frac{\cos(\Delta kh)}{2} \right]$$

$$= - \frac{1}{2n}\left[\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh)\right]$$

2.3.3 NBS approach

The expression for n+1 samples using the NBS approach [12] is:

$$Ay_j = \frac{1}{n + \Delta}\left(\sum_{j=0}^{n-1} y_j + \Delta * y_n \right) \qquad (2.17)$$

The corresponding error coefficient relating to the cosine term of the error expression is:

$$\Rightarrow \quad \alpha_k^S = A \sin(jkh)$$

$$= \frac{1}{n + \Delta}\left(\cot\left(\frac{k\,h}{2}\right) \sin^2\left(\frac{\Delta kh}{2}\right) + \frac{\sin(\Delta kh)}{2} \right.$$

$$\left. + \Delta\sin(nkh) \right)$$

$$= \frac{1}{n+\Delta}\left[\cot\left(\frac{k\,h}{2}\right)\sin^2\left(\frac{\Delta kh}{2}\right) + \frac{\sin(\Delta kh)}{2} - \Delta\sin(\Delta kh)\right]$$

$$= \frac{1}{n+\Delta}\left[\cot\left(\frac{k\,h}{2}\right)\sin^2\left(\frac{\Delta kh}{2}\right) + \frac{1}{2}(1-2*\Delta)\sin(\Delta kh)\right]$$

The corresponding error coefficient relating to the cosine term of the error expression is:

$$\Rightarrow \beta_k^S = A\cos(jkh)$$

$$= \frac{1}{n+\Delta}\left[\sin^2\left(\frac{\Delta kh}{2}\right) - \frac{1}{2}\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh) + \Delta\cos(nkh)\right]$$

$$= \frac{1}{n+\Delta}\left[\sin^2\left(\frac{\Delta kh}{2}\right) + \Delta\cos(\Delta kh) - \frac{1}{2}\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh)\right]$$

2.3.4. Modified Trapezoidal Approach

The expression for n+1 samples for an approach proposed by Zu-Liang [11] and referred to in the Modified

Trapezoidal Method (MTM) is:

$$Ay_j = \frac{1}{n+\Delta}\left[0.5y_0 + \sum_{j=1}^{n-1} y_j + 0.5y_n + \frac{\Delta}{2}(y_n + y_{n+\Delta})\right] \quad (2.18)$$

$$= \frac{1}{n+\Delta}\left[\sum_{j=1}^{n-1} y_j + 0.5(1+\Delta)(y_0+y_n)\right] \quad (2.19)$$

since $y_0 = y_{n+\Delta}$ for a measurement interval covering of an integral number of signal cycles.

The corresponding error coefficient relating to the sine term of the error expression is:

$$\alpha_k^N = A \sin(jkh)$$

$$= \frac{1}{n+\Delta}\left[\cot\left(\frac{kh}{2}\right)\sin^2\left(\frac{\Delta kh}{2}\right) - \frac{\Delta}{2}(\sin(nkh) + \sin(0))\right]$$

$$= \frac{1}{n+\Delta}\left[\cot\left(\frac{kh}{2}\right)\sin^2\left(\frac{\Delta kh}{2}\right) - \frac{\Delta}{2}\sin(\Delta kh)\right]$$

The corresponding error coefficient relating to the cosine term of the error expression is:

$$\beta_k^N = A\cos(jkh)$$

$$= \frac{1}{n+\Delta}\left[-\frac{1}{2}\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh) + \frac{\Delta}{2}\left(\cos(nkh) + \cos(0)\right)\right]$$

$$= \frac{1}{n+\Delta}\left[-\frac{1}{2}\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh) + \frac{\Delta}{2}\left(\cos(\Delta kh) + 1\right)\right]$$

$$= \frac{1}{n+\Delta}\left[-\frac{1}{2}\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh) + \Delta\cos^2\left(\frac{\Delta kh}{2}\right)\right]$$

$$= \frac{1}{n+\Delta}\left[\Delta\cos^2\left(\frac{\Delta kh}{2}\right) - \frac{1}{2}\cot\left(\frac{k\,h}{2}\right)\sin(\Delta kh)\right]$$

2.4 Determination of delta

Delta (Δ) is determined by employing a hardware scheme which resolves each sampling interval into 16 discrete sub-intervals.The sub-interval in which the end of a period occurs, is divided by 16 to obtain delta. Delta can thus obtained to the nearest one-sixteenth .

This scheme results in a more precise estimate of delta as opposed to the interpolation schemes suggested in the implementation of precision digital sampling meters [12] [14].

In order to minimize the errors resulting from a

deviation of the computed interpolated value Δ and the actual value of Δ, the sample size is adjusted so that $|\Delta| \leq 0.5$.

2.5 Computation of the desired parameters

The DC parameters of voltage, current and power are computed using the trapezoidal rule, while the corresponding AC parameters are computed using the processing method outlined in section 2.3.4.

The relationship between the parameters required, the value of y_j in equation 2.8 ,and the further computations required after applying the required processing method, are summarized below.

Parameter required	y_j	*Further computations required*
AC voltage	$(v_j)^2$	compute square-root
AC current	$(i_j)^2$	compute square-root
AC power	$v_j i_j$	
DC voltage	v_j	
DC current	i_j	
DC power	$v_j i_j$	

v_j and i_j are the respective sampled values of voltage and current.

Due to the use of fixed point arithmetic for the computation of most of the AC parameters,the following modified formula is used.

$$Ay_j = \frac{1}{16n + 16\Delta}\left(16\sum_{j=1}^{n-1} y_j + 0.5(16+16\Delta)(y_0+y_n) \right) \quad (2.20)$$

The above equation is used in the determination of power.

The determination of the rms values however, includes a correction for any dc terms present. The resulting expression is thus;

$$\text{RMS value} = \sqrt{A(y_j^2) - (Ay_j)^2} \quad (2.21)$$

where y_j is v_j or i_j depending on the parameter required.

and **A** is the processing method used

Another parameter, the power factor (P.F) [11] for sinusoidal signals, is determined as follows:

a) A Single-phase system

$$P.F. = \frac{\text{POWER}}{V_{jm} I_{jm}} \quad (2.22)$$

b) A Balanced three-phase three-wire system

$$P.F. = \left[\frac{(P1 + P2)^2}{(P1+P2)^2 + 3(P2-P1)^2} \right]^{\frac{1}{2}} \quad (2.23)$$

where v_{jm} is the maximum sampled voltage value

i_{jm} is the maximum sampled current value

P1 and P2 are the phase powers,with P2 being the larger value.

2.6 Summary of results

The relationships obtained for the various processing methods in section 2.3.4., are compared using the relevant parameters employed in the practical implementation of the meter. The error values thus obtained would have been obtained if error-free analog input devices as well as a perfect analog-to-digital conversion process occurred in the meter.

From the equations obtained in section 2.3.4, it is observed that the error coefficients are functions of Δ(delta), k (the harmonic number), h (the sampling interval in radians) and n the number of samples processed. In the actual meter implementation, a fixed sampling interval of 32μs and an acquisition cycle of one period of the input signal cycle are used. In addition, for the processing method used, the sample size is adjusted to ensure $|\Delta| \leq 0.5$. Frequencies around the nominal power line frequency of 60 hz which result in values of delta which satisfy the following relation are used.

$$(0 < |\Delta| \leq 0.5)$$

The frequencies employed range from 59.925 hz to 60.035 hz

in increments of 0.005 hz.

Graphical representations of the results deduced in section 2.3.4 and the relationships between the various processing method and the approach employed, are indicated in figs. 2.1 - 2.12. The 32μs sampling interval used in the instrument is employed for these representations.

Figs. 2.1 - 2.8 are plotted for signal harmonics of 1 (the fundamental), 2, 10 and 20. In figs. 2.9 - 2.12, harmonics from 1 to the number of samples processed in a cycle (521 in the cases analyzed) are used. Figs. 2.1 - 2.6 indicate the relationship between the ratios of the error coefficients of the average, trapezoidal and NBS approach to that of the procedure used in this work. Figs 2.1 - 2.3 refer to the alpha ratios while figs. 2.4 - 2.6 are for the beta ratios. Figs 2.7 and 2.8 indicate the relationship between the values of error coefficients α_k^N and β_k^N respectively for signal harmonics of 1 (fundamental), 2, 10 and 20.

Figs. 2.1 - 2.5 indicate a general decrease in the error ratios with increasing harmonic number. The processing method used in this work results in an improvement of the alpha error coefficient of the signal fundamental frequency component by a factor of about 10^5 compared with the component calculated from the average, trapezoidal and NBS methods. The beta error coefficients that result from using the average and trapezoidal methods are identical, with an improvement of about 10^5 being

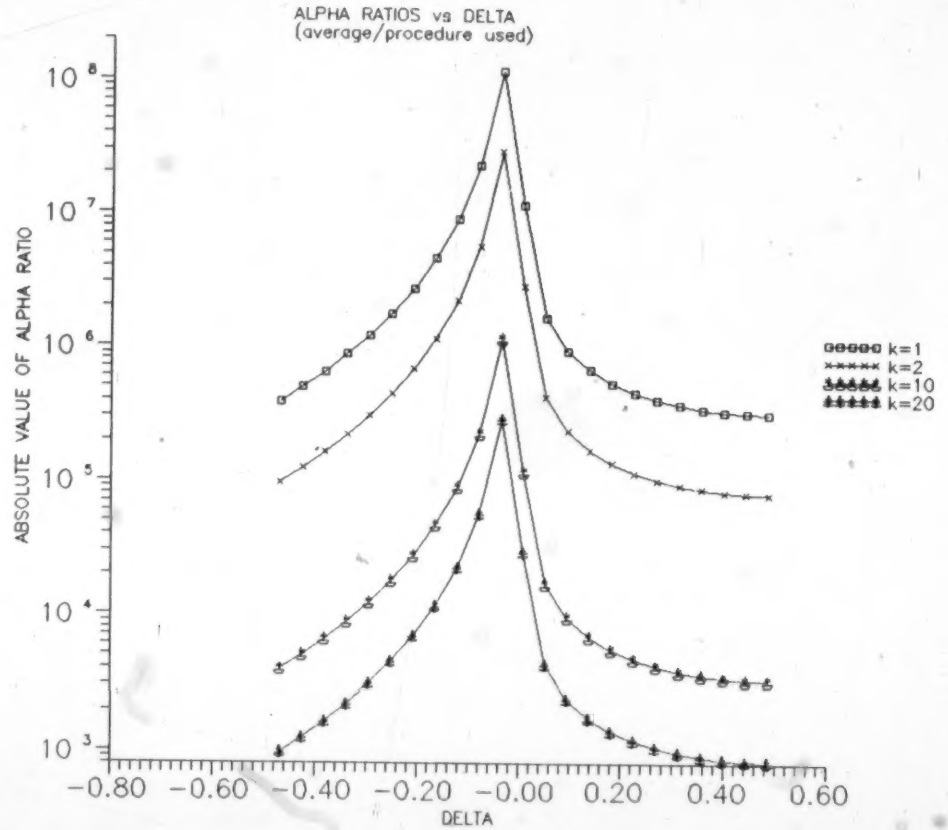


FIG. 2.1 GRAPH OF ALPHA RATIOS vs DELTA (AVERAGE/APPROACH USED)

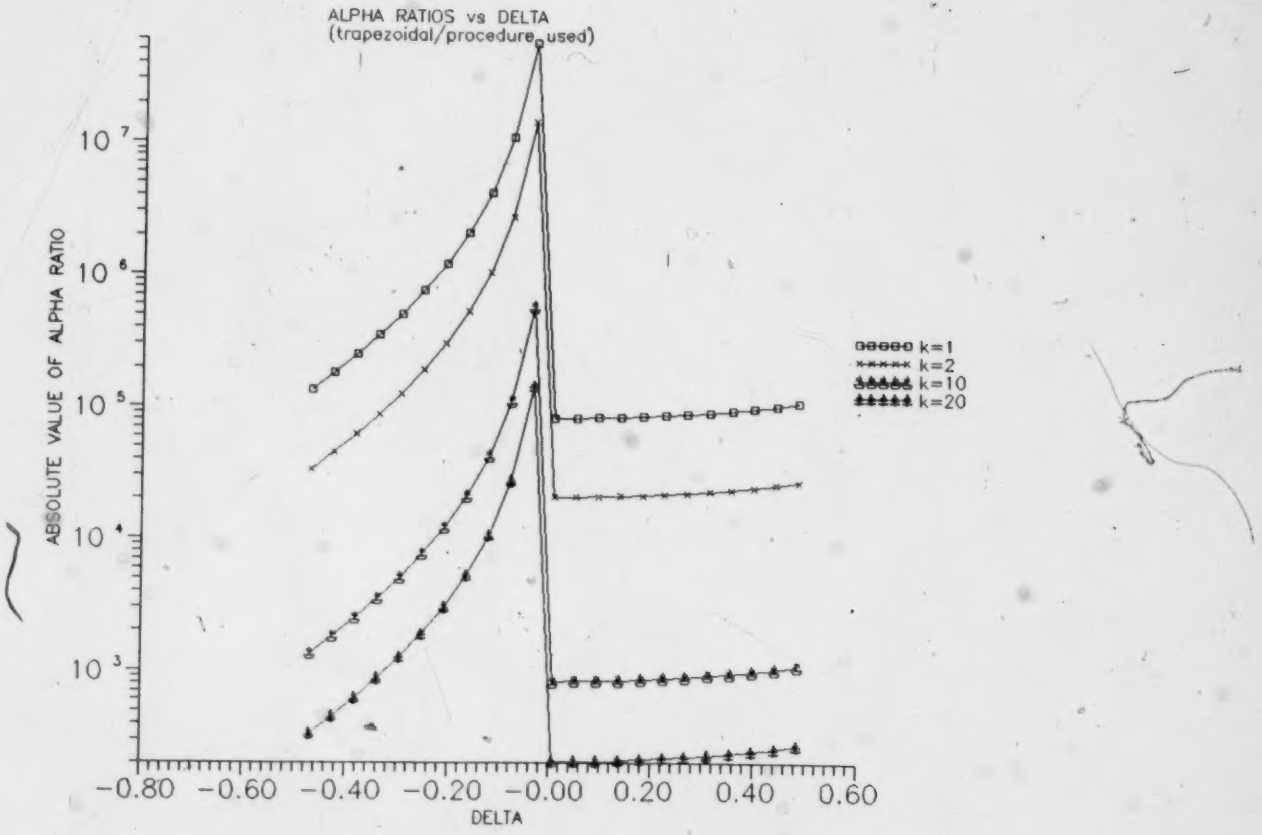


FIG. 2.2 GRAPH OF ALPHA RATIOS vs DELTA (TRAPEZOIDAL/PROCEDURE USED)

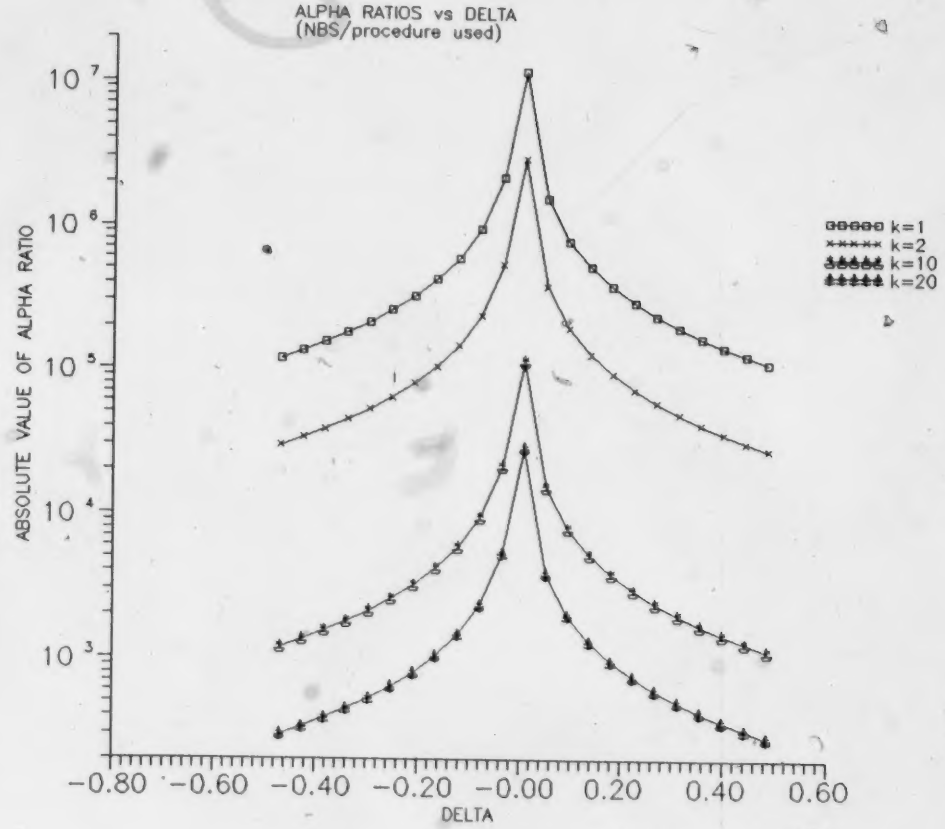


FIG. 2.3 GRAPH OF ALPHA RATIOS vs DELTA (NBS/PROCEDURE USED)

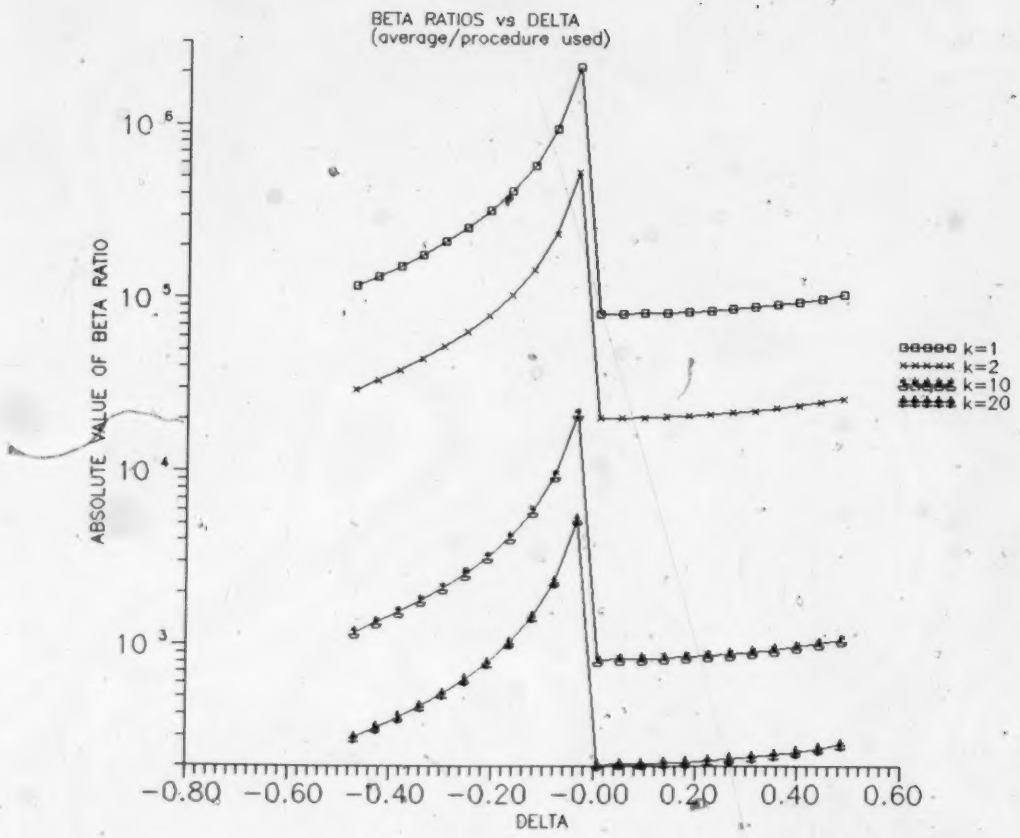


FIG. 2.4 GRAPH OF BETA RATIOS vs DELTA (AVERAGE/PROCEDURE USED)

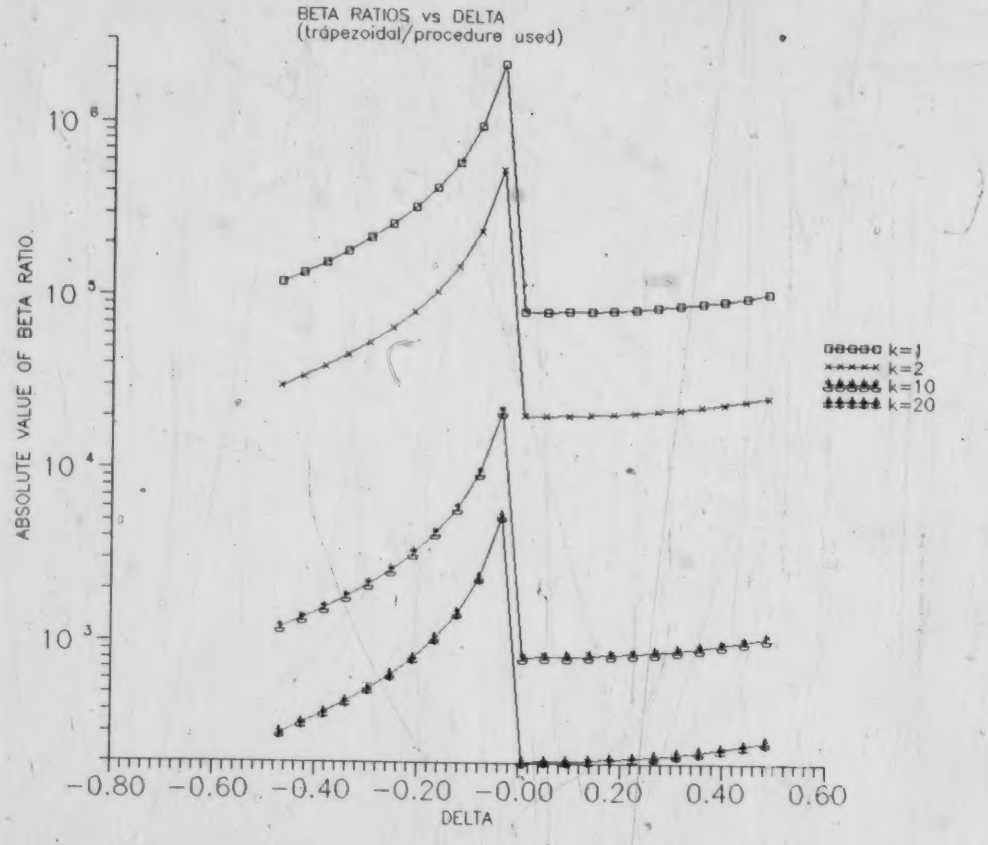


FIG. 2.5 GRAPH OF BETA RATIOS vs DELTA (TRAPEZOIDAL/PROCEDURE USED)

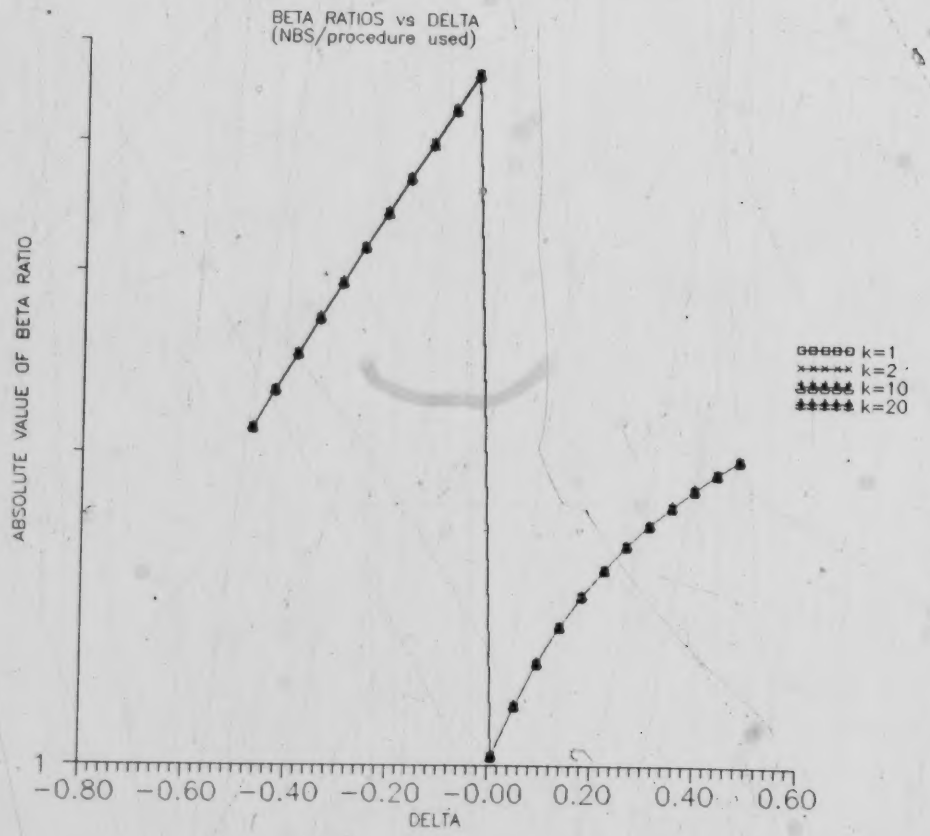


FIG. 2.6 GRAPH OF BETA RATIOS vs DELTA (NBS/PROCEDURE USED)

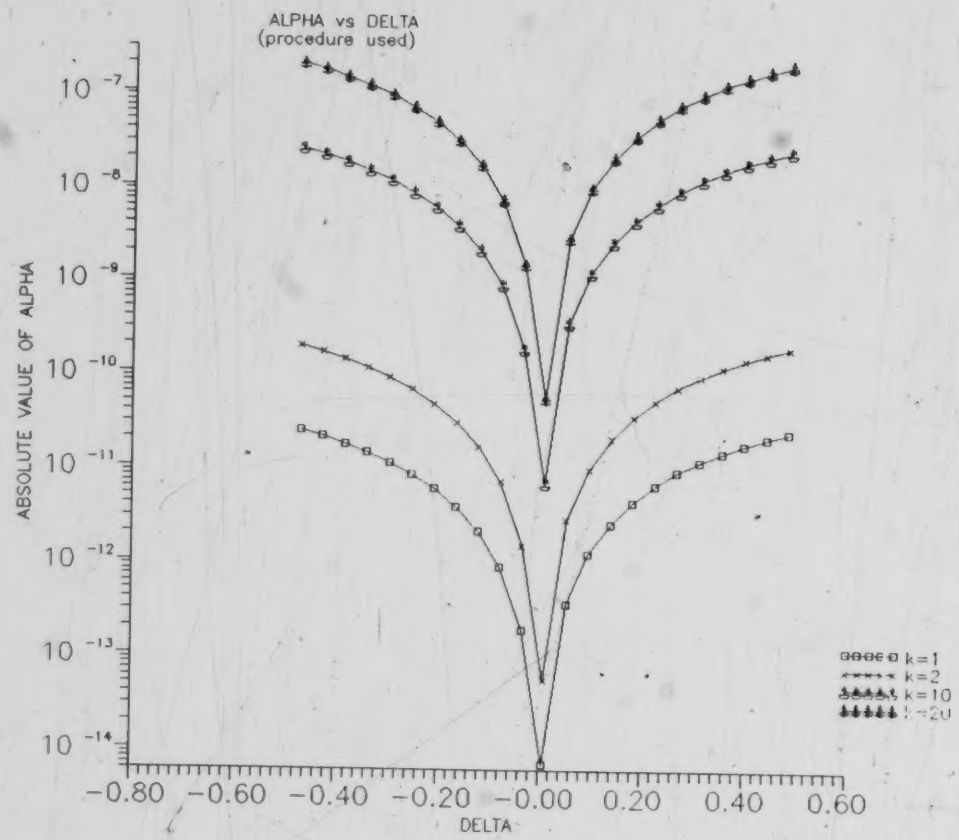


FIG. 2.7 GRAPH OF ALPHA vs DELTA FOR THE PROCEDURE USED (MTM)

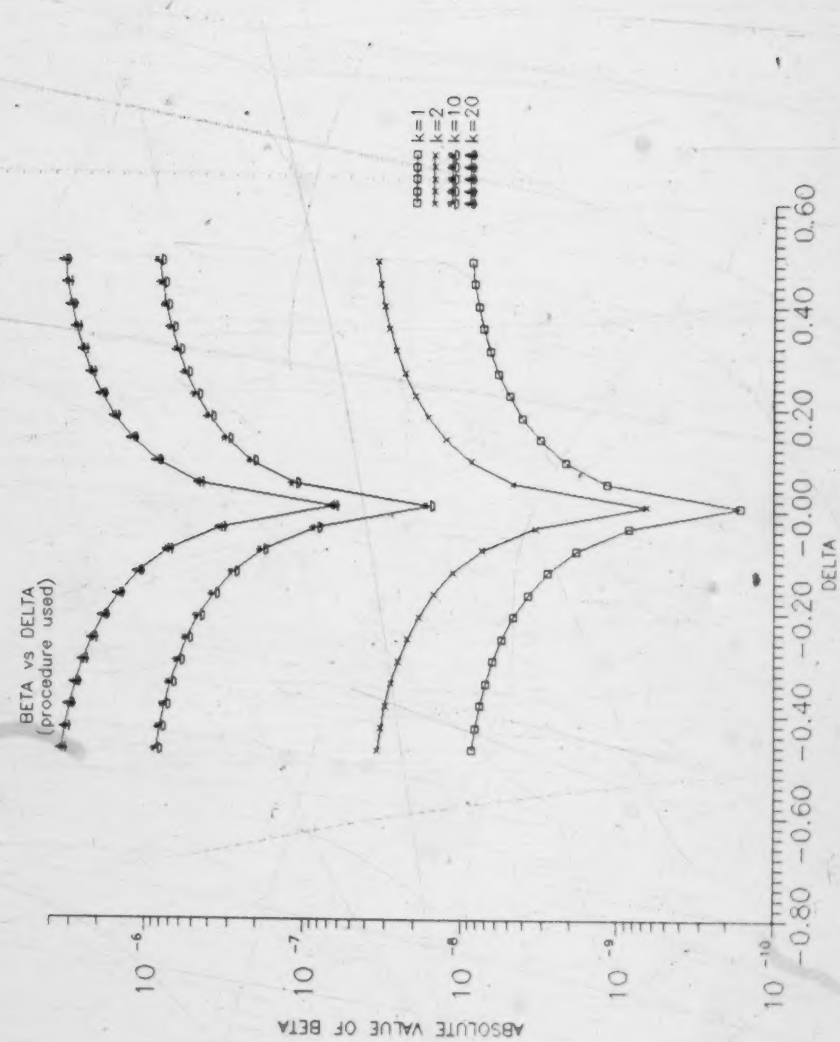


FIG. 2.8 GRAPH OF BETA vs DELTA FOR THE PROCEDURE USED (MTM)

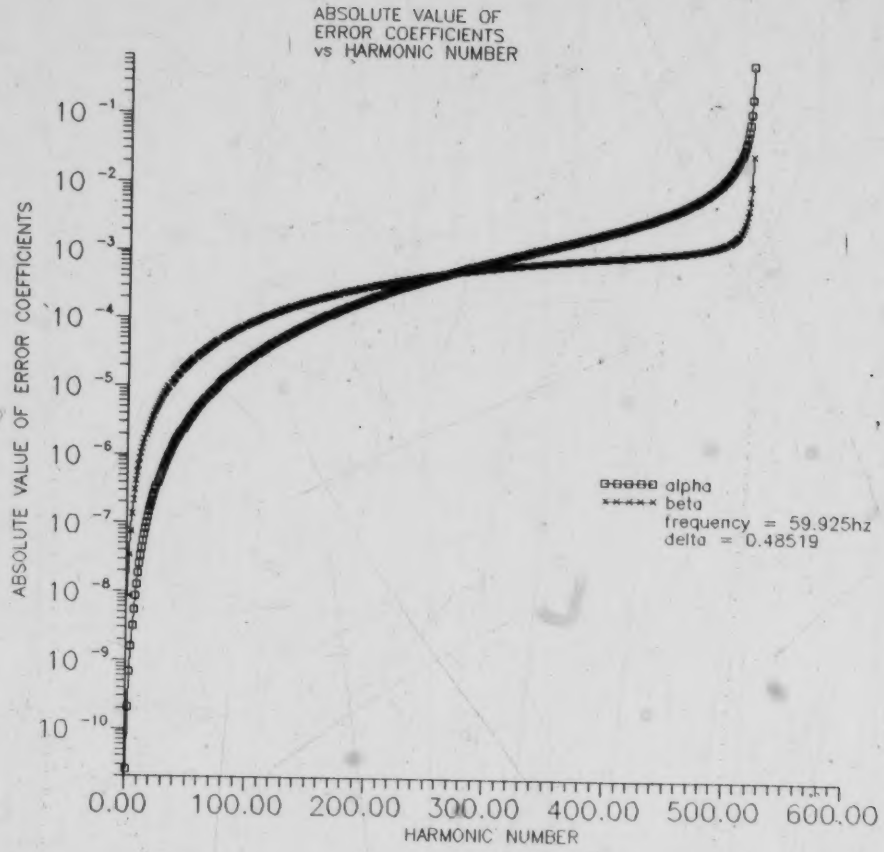


FIG. 2.9 GRAPH OF ABSOLUTE VALUE OF ERROR COEFFICIENTS vs HARMONIC NUMBER FOR A FREQUENCY OF 59.925 hz

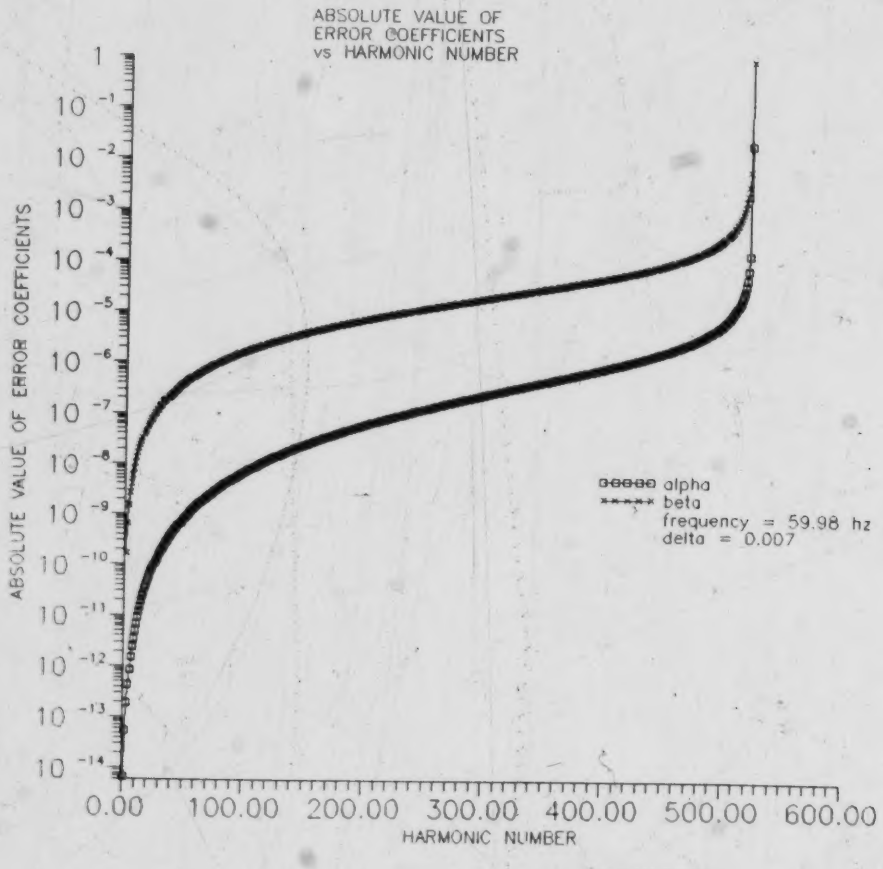


FIG. 2.10 GRAPH OF ABSOLUTE VALUE OF ERROR COEFFICIENTS vs HARMONIC NUMBER FOR A FREQUENCY OF 59.98 hz

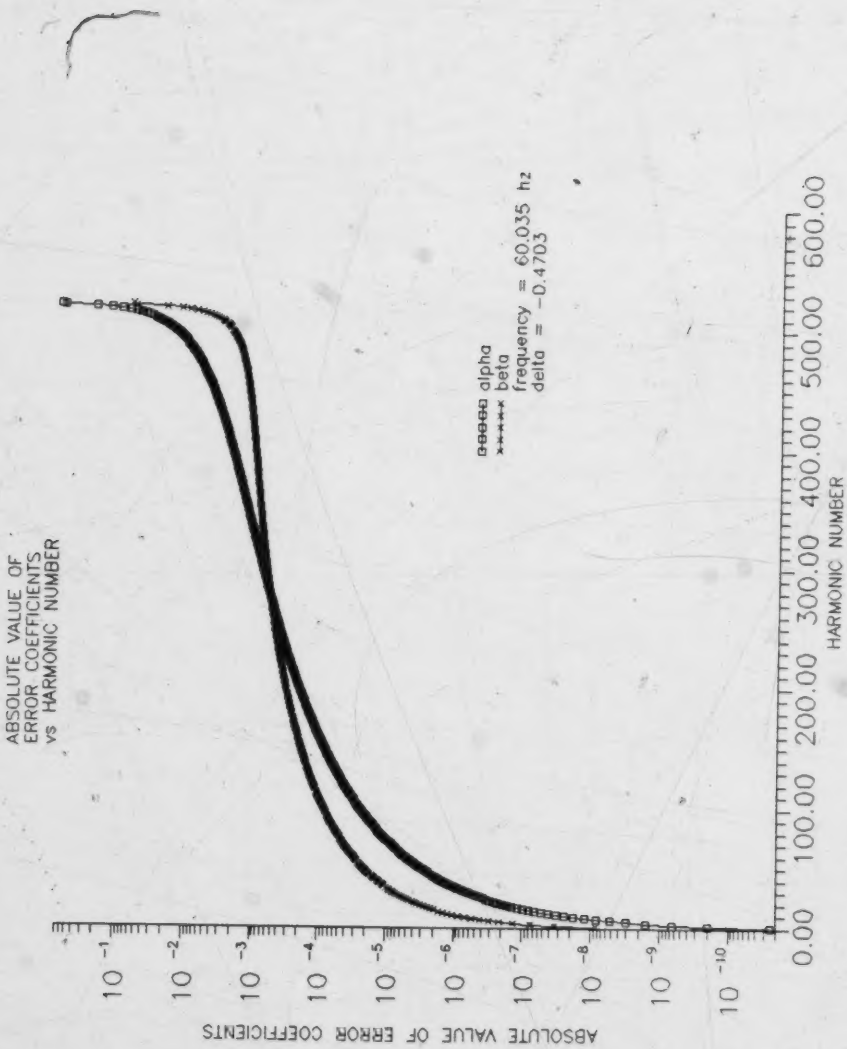


FIG. 2.11 GRAPH OF ABSOLUTE VALUE OF ERROR COEFFICIENTS vs HARMONIC NUMBER FOR A FREQUENCY OF 60.035 hz

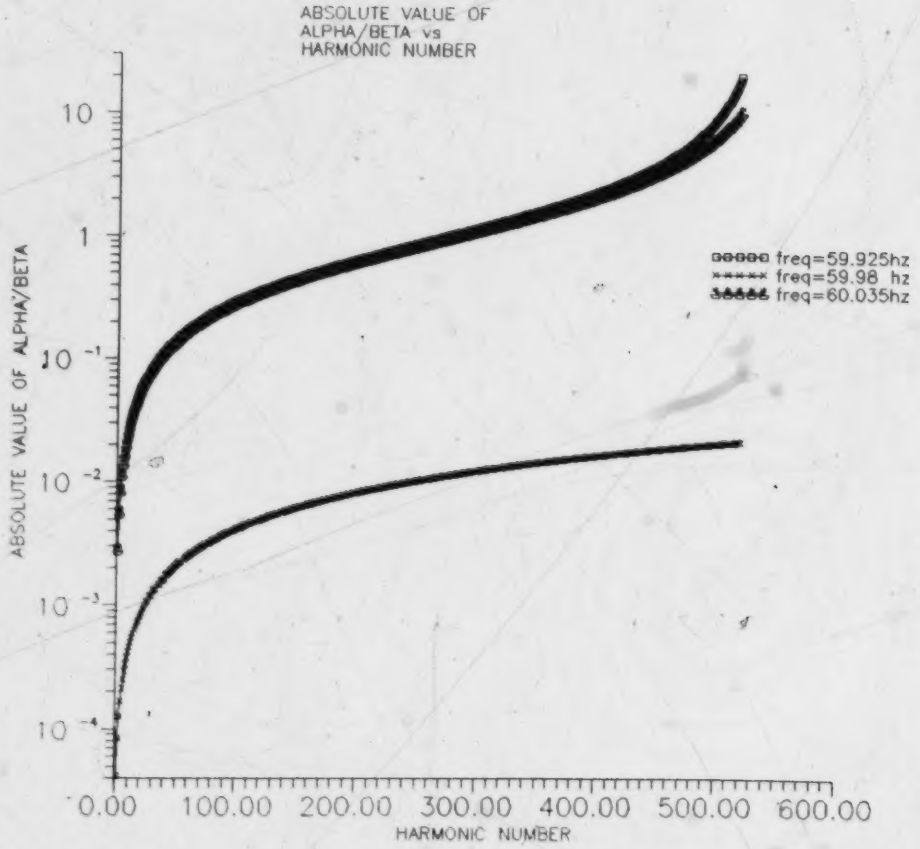


FIG. 2.12 GRAPH OF ABSOLUTE VALUE OF THE RATIO ALPHA/BETA vs HARMONIC NUMBER

obtained with the processing method used in this work. From fig. 2.6, it is observed that the beta error ratios are approximately equal irrespective of the order of the harmonic. Figs. 2.9 - 2.12 relate to the processing method used. The absolute values of alpha and beta (figs. 2.9 - 2.11) as well as their ratios (fig. 2.12), are obtained for harmonics from 1 to 521. The choice of frequencies of 59.925 hz, 59.98hz and 60.035 hz was made because they result in delta values close to 0.5, 0 and -0.5 respectively. Fig 2.12 indicates the absolute values of the ratios of alpha and beta obtained in figs. 2.9 - 2.11.

In determining the error contribution to a meter reading for a particular harmonic term, the worst case contribution will correspond to the largest value of either alpha or beta at that harmonic. As outlined previously, if the starting point of the acquisition cycle could be varied, then in principle the error term due to a particular harmonic could be minimized.

3. DIGITAL SAMPLING WATTMETER HARDWARE

The choice of the system hardware is governed by the following design considerations:

(a) A highly accurate system using off-the-shelf components.

(b) Adequate speed in handling 12-bit values and the mathematical manipulations to be performed on the latter.

An 8 Mhz zero-wait state 68000 microprocessor based system is employed for the current design. The instruction set of this processor incorporates various data manipulation instructions suitable for the mathematical computations involved. The hardware also facilitates the incorporation of debugging support which simplifies system maintenance.

A block diagram of the system hardware is shown in Fig. 3.1 (page 46). The system sub-units are; the CPU module, the I/O module, the timing controller module, the analog-to-digital interface module and the input-conditioning module. A more detailed block diagram showing the interconnecting signal lines are given in Fig. A.1.

3.1 CPU module

The block diagram of this module is shown in Fig. 3.2 (page 46). This circuit consists of the following :

a) the microprocessor circuit,

b) the memory circuit,

c) the interrupt and watchdog timer circuitry.

A more detailed block diagram showing the control signal interconnections is shown in the appendix as Fig. A.2.

3.1.1 Microprocessor circuitry

The block diagram of this circuit is shown in Fig. 3.3 (page 47). The heart of this circuitry is a 68000 microprocessor clocked by an 8 Mhz system clock [9]. The reset circuitry generates the required reset signal on power-up or following the assertion of the externally provided reset switch.

The system memory map shown in Fig. 3.4 (page 48), indicates a partial decoding of the 16 megabyte direct address space. This scheme results in a simple implementation of the decoding scheme required for the microprocessor peripheral devices. The I/O and A/board address spaces are further decoded as indicated in Fig. 3.5 (page 49).

The DTACK and VPA generation circuitry generate the required data transfer acknowledgement signals for asynchronous and synchronous data transfers respectively, between the microprocessor and external devices.

Data, address and control line buffers are provided for communication external to the microprocessor circuitry.

The detailed schematic of this circuitry is shown in

Fig. A.3 and Fig. A.4.

3.1.2 Memory circuitry

The block diagram of this circuitry is shown in Fig. 3.6 (page 50). The 16-bit data bus of the microprocessor necessitates the use of a pair of byte-wide memory devices. Two 8 kB byte-wide EPROMs having a maximum access time of 200ns, as well as a pair of 32 kB byte-wide static RAMs having a maximum access time of 150ns are used. The EPROMs store the program code and permanent data. The latter devices are decoded at a base address of 000000 hex. This scheme results in a simplified reset and interrupt circuitry thus further simplifying the system design. The static rams are decoded at a base address of 200000 hex. The RAMS are employed for the storage of sampled data values as well as temporary variables. 8k words are allocated for the storage of the sampled data values. The detailed circuitry of this sub-unit is indicated in Fig. A.5.

3.1.3 Interrupt and Watchdog timer circuitry

The block diagram of this circuitry is shown in Fig. 3.7 (page 51). This circuitry encodes the interrupt signals and also generates the required interrupt acknowledgement signals. The interrupt encoding scheme results in the following priority scheme for the servicing of interrupts:

the GPIA circuitry, the keyboard controller circuitry and the PTM circuitry.

The watchdog timer asserts the microprocessor bus error line following the absence of a data transfer acknowledge signal. The assertion of the bus error line occurs 40 system clock cycles following the start of a data transfer sequence.

The detailed circuit diagram is shown in Fig. A.6.

3.2 I/O module

This section consists of five individual subunits as indicated in the block diagram of Fig. 3.8 (page 52). The microprocessor peripherals in this section interface with the 68000 microprocessor via the synchronous control signal lines. A more detailed block diagram showing the control signal interconnections is shown in the appendix as Fig. A.7.

The PTM subunit is centered around the 6840 programmable timer module. This unit serves two main functions. It is employed in the detection of the presence of an AC or DC signal on the trigger input line. This simple scheme enables the use of an interrupt approach through a device geared towards the efficient generation of interrupts. Secondly, the timer module is employed in the determination of the trigger input line signal frequency. The detailed circuit diagram of this circuit is

shown in Fig. A.8.

The PIA circuitry is centered around the 6821 peripheral interface adapter IC. Both peripheral ports of the latter device are configured as outputs. In essence, this circuitry functions as a microprocessor controlled output latch. The detailed circuit diagram of this circuitry is shown in Fig. A.9

The keyboard circuitry is centered around the Intel 8279 programmable keyboard and display controller IC [7]. The main function of this circuitry is to provide the necessary interface between the microprocessor and the input terminal. This circuit also decodes and debounces the keypad switches. The 4 by 4 keypad matrix and a single pole double throw switch result in 32 unique keypad combinations for the various wattmeter functions. An interrupt driven scheme is employed in determining a key closure. The detailed circuit diagram of this section is shown in Fig. A.10.

The display circuitry uses 2 devices for display of the program results as well as the system status. The primary unit is a 4 line by 20 column LCD display. In addition, a 10 segment tri-colour bargraph is also used. The various segments of the latter device being labelled 0 to 9. The LCD display is connected to the microprocessor as a memory mapped device at address 6C0000 hex. The LCD display unit includes an onboard character generator ROM. The latter

facility results in simple LCD display routines. The LCD unit also performs the refreshing of the display thus freeing up the microprocessor for other tasks. A further advantage in using this device is the possibility of writing to any of the 80 locations on the screen independently of the others. The 4 line LCD places a limitation on the number of items that can be displayed on the screen at a time. The bargraph display segments indicate the system as well as the program status. Enabling or disabling individual display segments is effected through the PIA. The detailed circuit diagram of this section is shown if Fig. A.11.

The GPIA circuitry provides the interface required to implement the IEEE-488 bus (GPIB) protocol. The heart of this subunit, the Motorola MC68488 GPIA IC, facilitates the implementation of the bus protocol. This circuitry together with the associated software enables the instrument to implement the following IEEE-488 standard features: talker, listener, complete source and acceptor handshake, serial and parallel poll, device trigger, device clear and the remote/local procedure. The detailed circuit diagram of this circuitry is shown in Fig. A.12.

3.3 Timing Controller module

The block diagram of this module is shown in Fig. 3.9 (page 53). This subunit comprises the conversion timing

logic, the AC detection logic and the acquisition state logic. A more detailed block diagram showing the control signal interconnections is shown in the appendix as Fig. A.13.

The conversion timing logic produces the signals required for the analog-to-digital (A/D) converters and the sample-and-hold amplifiers. The use of Fairchild Advanced Schottky TTL (FAST) logic as well as the derivation of the various timing signals from the 8 Mhz system clock results in minimal jitter as well as precise timing signals. The A/D converter signals are the 1Mhz analog-to-digital (A/D) converters clock signal and the 2μs start of conversion signal. The primary sample-and-hold amplifier signal is that of the 32μs sampling interval. Some relevant timing signals generated by this logic and their relationship to the analog-to-digital interface circuitry is indicated in Fig. 3.10 (page 54).

The AC detection logic provides the automatic AC or DC mode detection required for the wattmeter operation.

The acquisition state logic provides the status signals employed by the software in determining the state of the current acquisition cycle. This circuit also determines the end correction required following an acquisition cycle involving an AC signal.

The detailed circuitry of this subunit is shown in Figs. A.14 and A.15.

3.4 Analog-to-Digital interface module

The block diagram of this module is shown in Fig. 3.11 (page 55). A more detailed block diagram showing the various signal interconnections is shown in the appendix as Fig. A.16.

The purpose of this section is to provide a digital equivalent of the analog input signal after the required input conditioning has been performed. This subunit comprises 4 similar channel interface sections representing each of the input channels. Each section has as its main devices, a voltage scaling and translation section, a sample-and-hold amplifier, a 12-bit analog-to-digital (A/D) converter, as well as an output latch for the temporary storage of the converted values. An input signal in the range -10v to +10v applied to the respective voltage scaling and translation section produces a 0-5v output. The latter voltage is applied to the corresponding sample-and-hold amplifier and subsequently converted to a 12-bit value in the respective A/D converter. The acquisition scheme employed, results in the sampling and conversion of signals on all 4 channels being performed simultaneously. The circuit logic is set up so that the results of the previous conversions are stored in the latches while the current conversion is taking place. This scheme results in 30μs of the 32μs sampling interval being available for use by the

microprocessor.

The detailed circuit diagram of this subunit is shown if Figs. A.17 - A.20.

3.5 Input-conditioning circuitry

Fig. 3.12 (page 56) shows the block diagram of this module. The four sections making up this module are; the Phase 1 circuitry, the Phase 2 circuitry, the analog switches circuitry and the trigger circuitry. A more detailed block diagram showing the various signal interconnections is shown in the appendix as Fig. A.21.

3.5.1 Phase 1 circuitry

The block diagram of the Phase 1 circuitry is shown in Fig. 3.13 (page 56). This section consists of a voltage circuitry and a current circuitry. The voltage circuitry also referred to as the channel 1 circuitry, provides the interface between the digital wattmeter and the input voltage source. This circuitry is employed for single phase measurements or as the first phase in three-phase system measurements. Two input voltage ranges of 200V and 350V peak are provided. Initial signal attenuation is provided by the resistive voltage divider stage. Further buffering and amplification to the -10v to +10v required for the subsequent A/D interface stage is provided by the amplifier and buffer stage [4]. The current circuitry also referred

to as the channel 2 circuitry, provides the interface between the digital wattmeter and the input current source. This circuitry is employed for single phase measurements or as the first phase in three-phase system measurements. A current shunt is employed for current measurement. The voltage generated across the shunt, is amplified by the subsequent amplifier and buffer stage circuitry to the -10v to +10v range required for the subsequent A/D interface stage. The detailed circuit diagram of this stage is shown in Fig. A.22.

3.5.2 Phase 2 circuitry

The block diagram of the Phase 2 circuitry is shown in Fig. 3.14 (page 57). This circuitry is identical to the Phase 1 circuitry and used in three-phase measurements or measurements of a second setup. This section comprises a voltage circuitry also referred to as the channel 3 circuitry and a current section referred to as the channel 4 circuitry. The detailed circuit diagram of this stage is shown in Fig. A.23.

3.5.3 Analog switches circuitry

The block diagram of this section is shown in Fig. 3.15 (page 57). This section consists of two main blocks; the trigger source switching circuitry and the system mode switch circuitry. Both switching circuits are controlled

through the PIA. The trigger source input selects the output of either channel 1 or 2 as the trigger source. Filtering of the output of this stage is provided by the low-pass filter before subsequent processing. The system mode switch selects the input to the A/D interface stage. Two options are available for the above-mentioned stage. These are ; the calibration terminal or the input-conditioning subunit. The analog switches are controlled by the 6821 PIA IC. This approach enables the selection of the specified sources under software control. The detailed circuit diagram of this circuit is shown in Fig. A.24.

3.5.4 Trigger circuitry

Fig. 3.16 (page 58) shows the block diagram of this circuitry. This circuit is basically that of a variable threshold voltage level detector. The triggering level adjust resistor provides the input switching threshold voltage of the voltage comparator. The comparator includes logic to generate a TTL compatible output signal. The detailed circuitry for this section is shown if Fig. A.25.

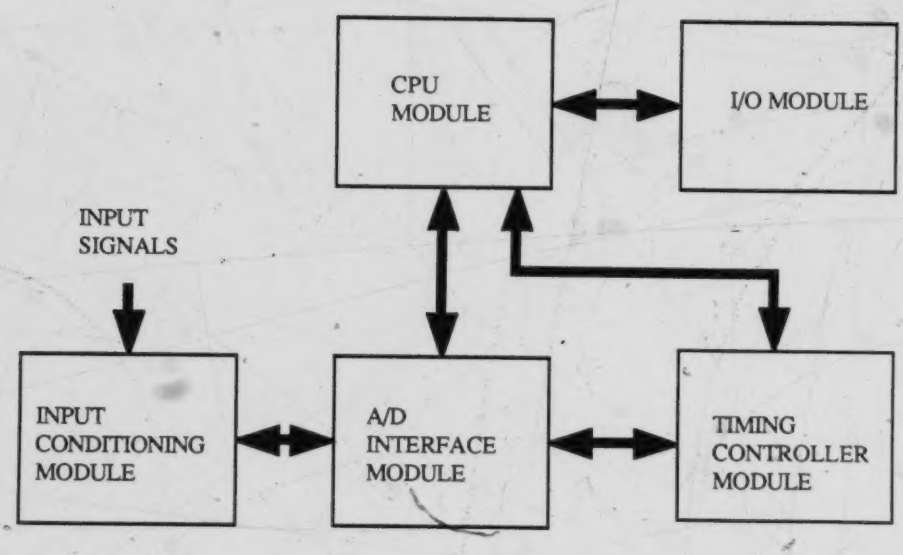


Fig. 3.1 Block diagram of wattmeter

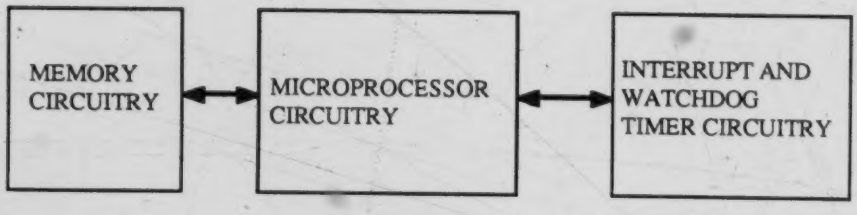


Fig. 3.2 Block diagram of CPU module

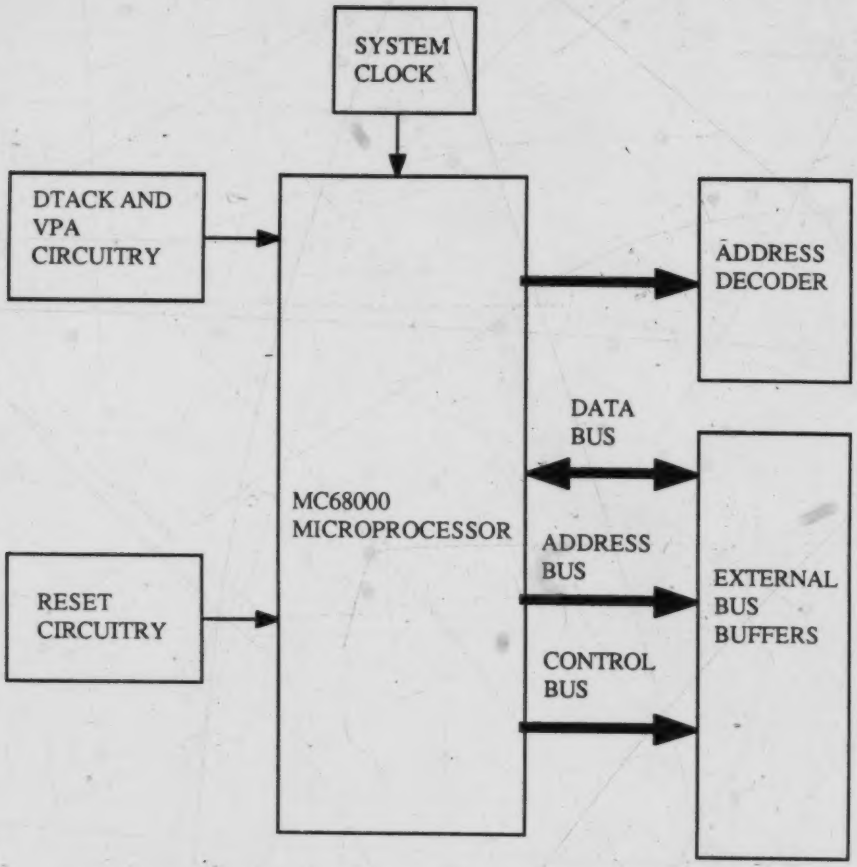


Fig. 3.3 Block diagram of microprocessor circuitry

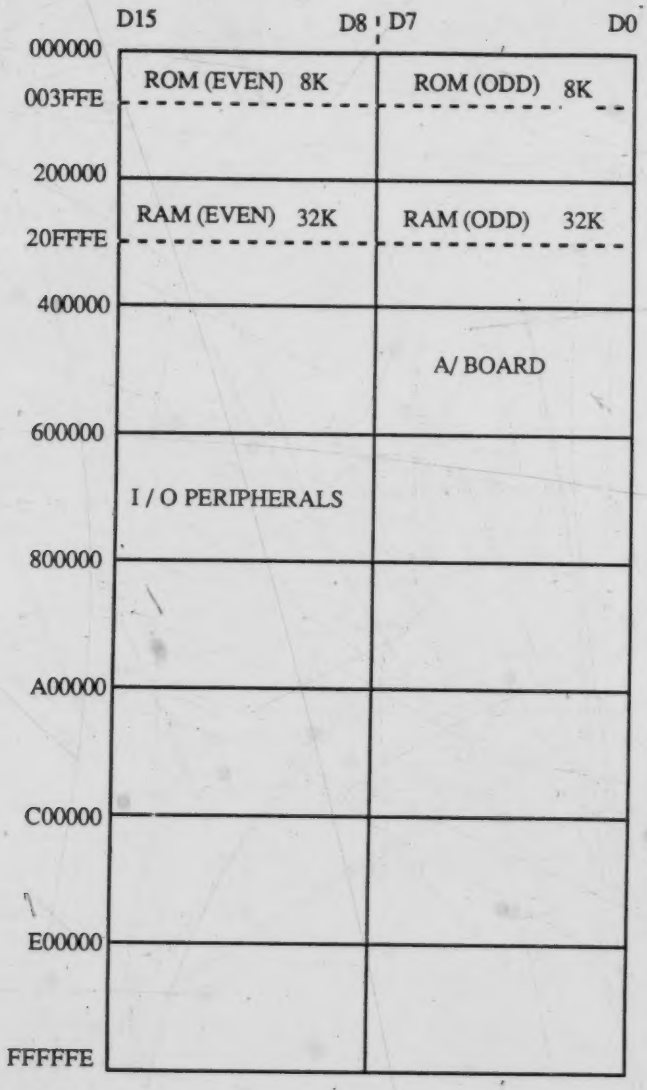


Fig. 3.4 System memory map

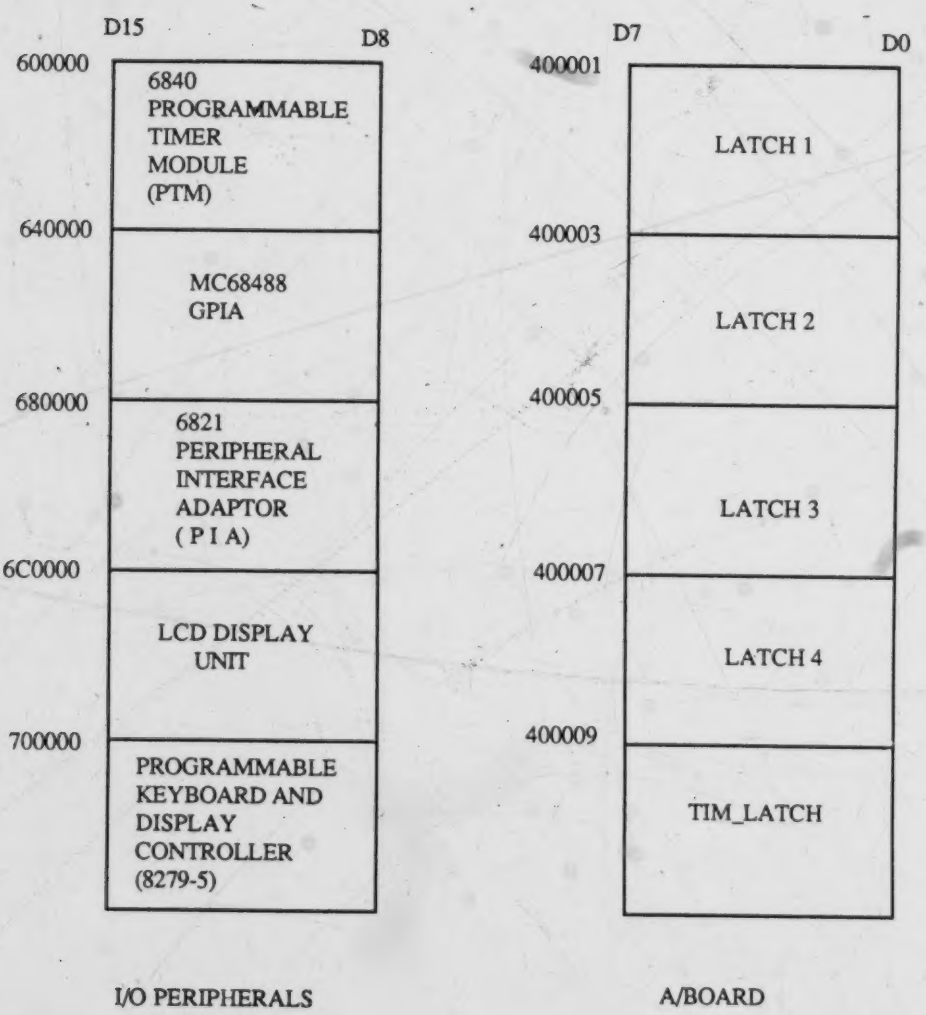


Fig. 3.5 Detailed memory map of I/O peripherals and A/Board

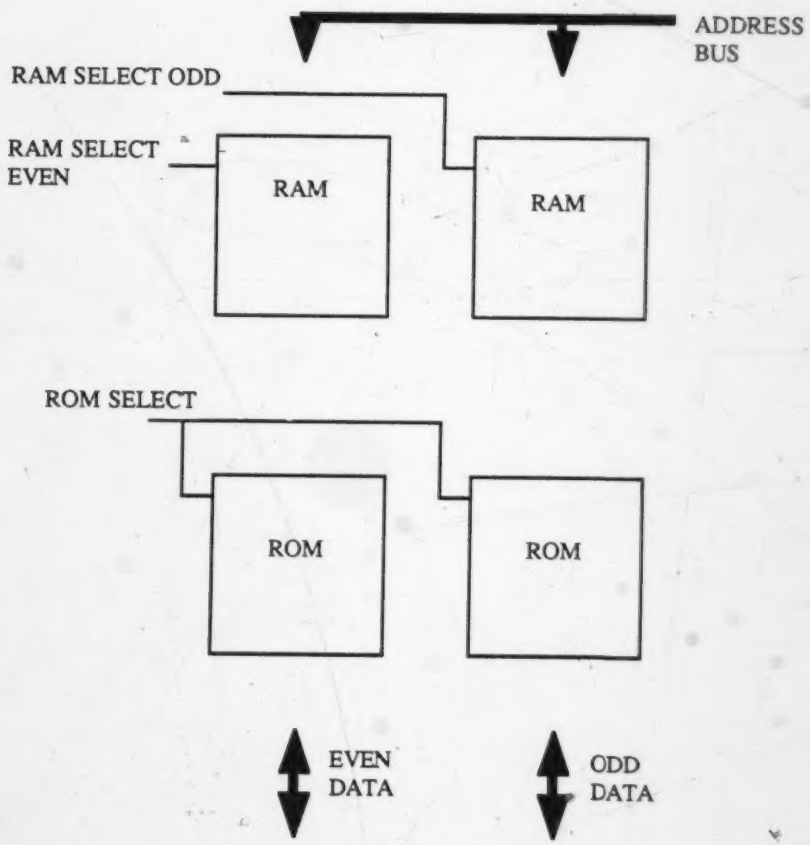


Fig. 3.6 Block diagram of memory circuitry

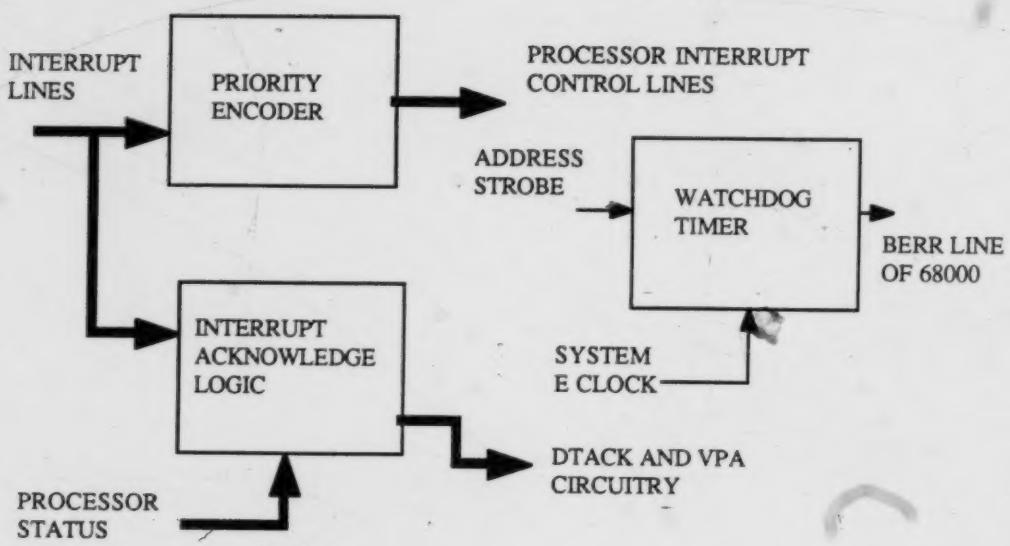


Fig. 3.7 Block diagram of interrupt and watchdog timer circuitry

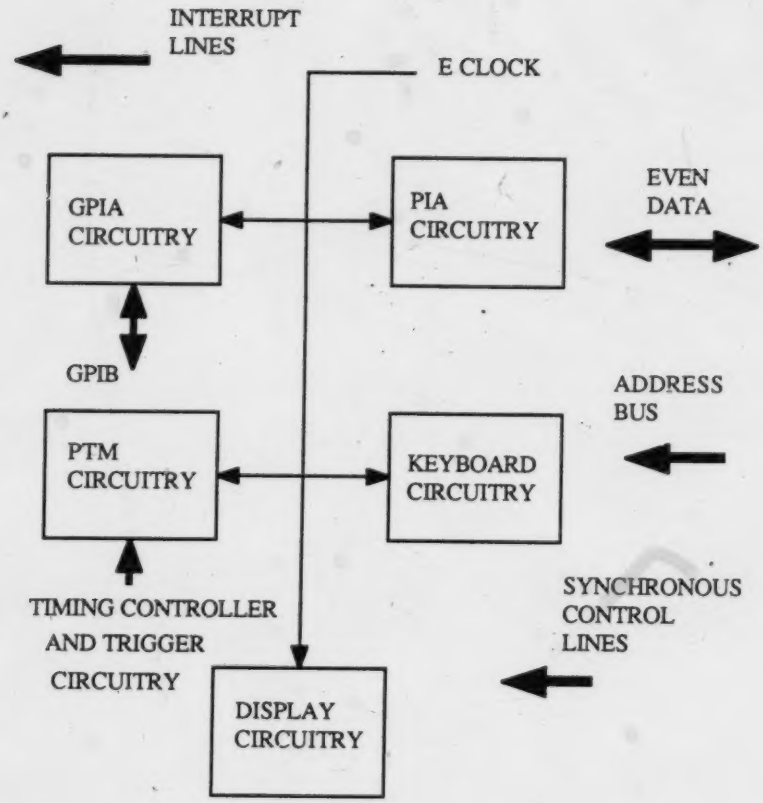


Fig. 3.8 Block diagram of I/O module

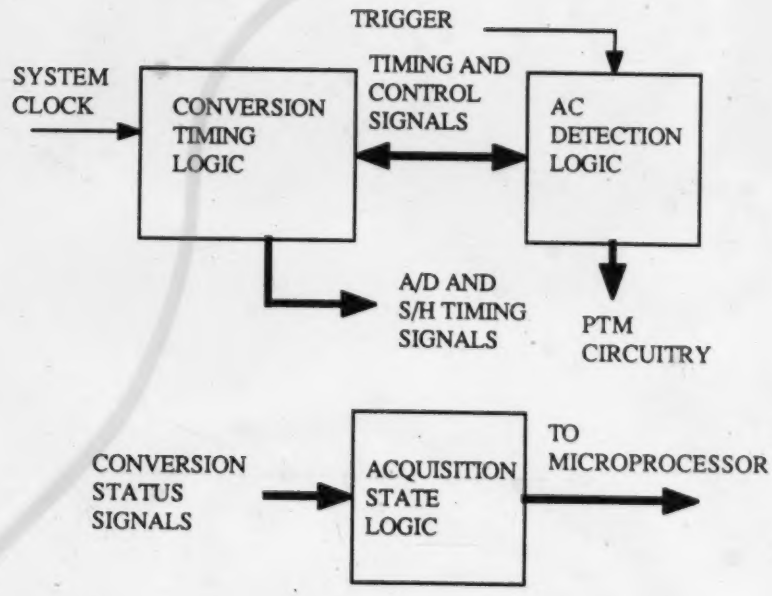


Fig. 3.9 Block diagram of the timing controller module

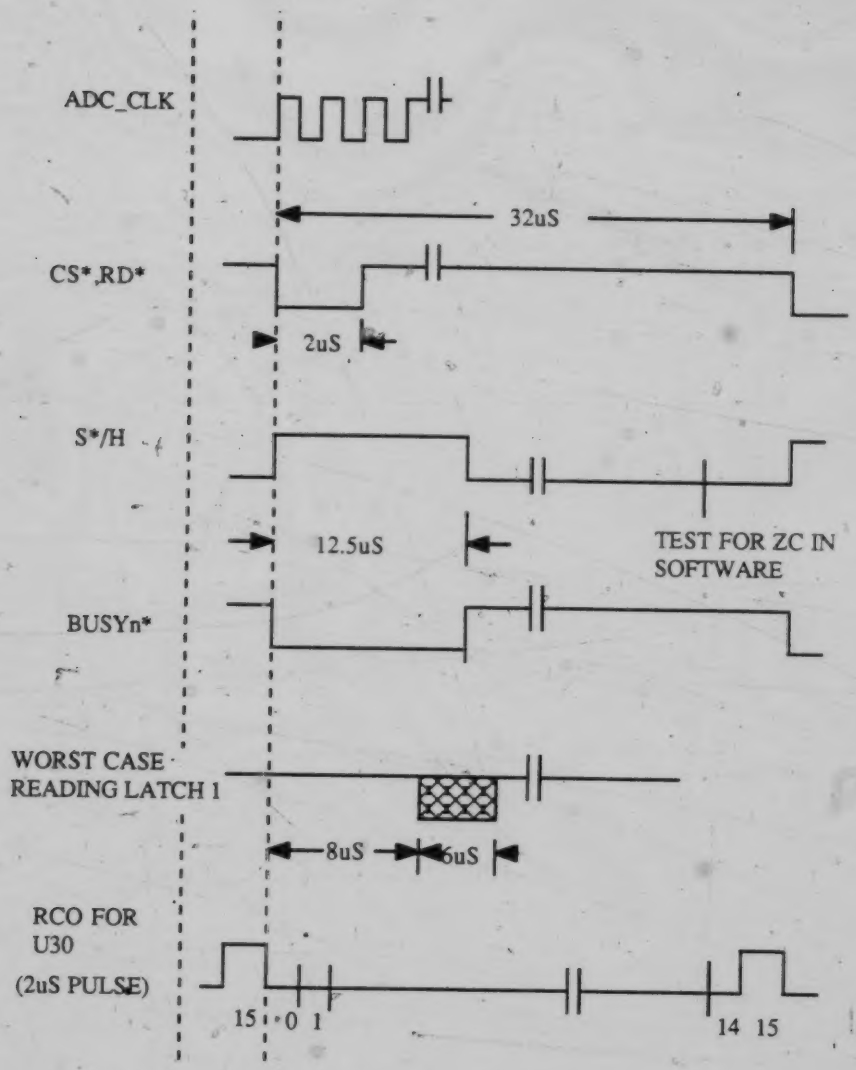


Fig. 3.10 A/D Interface timing diagram

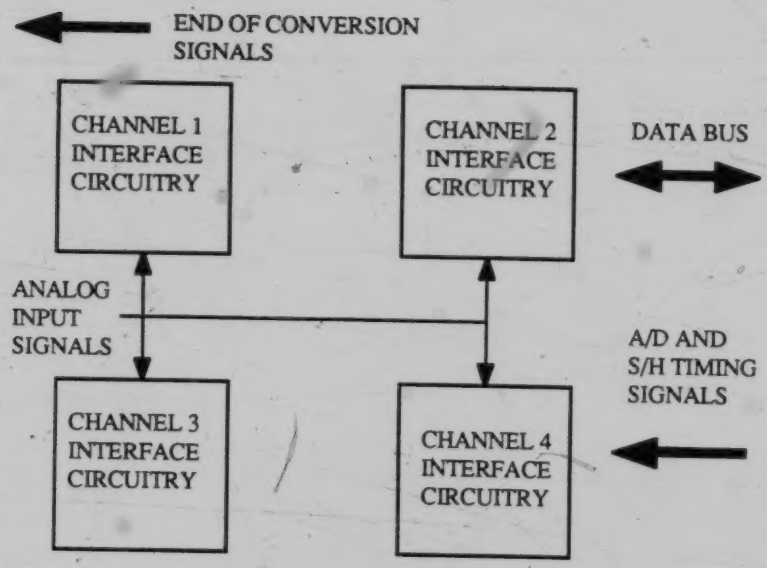


Fig. 3.11 Block diagram of the Analog-to-Digital interface module

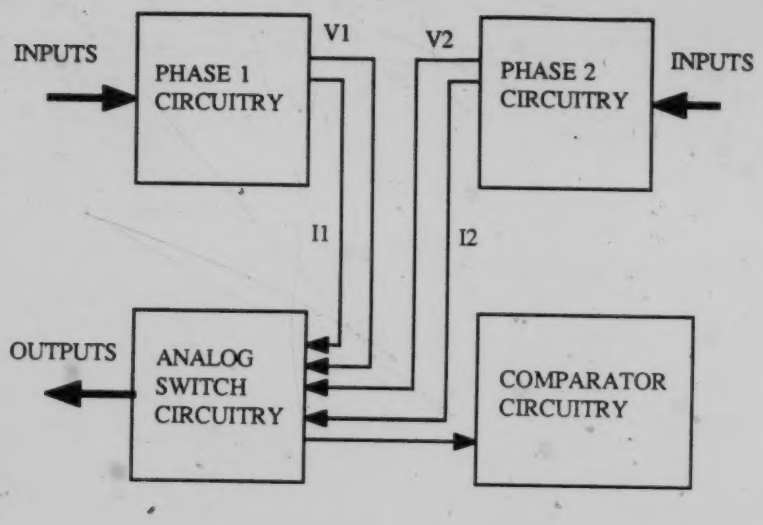


Fig. 3.12 Block diagram of the input conditioning module

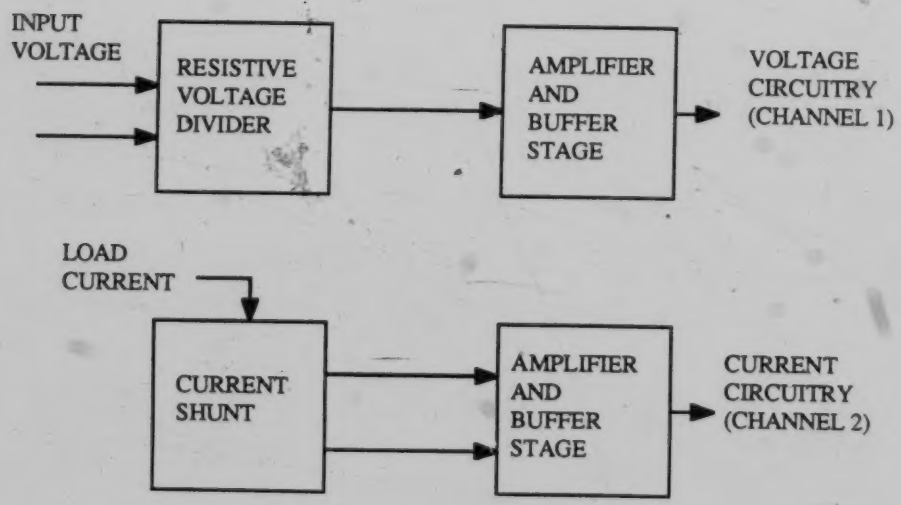


Fig. 3.13 Block diagram of the phase 1 circuitry

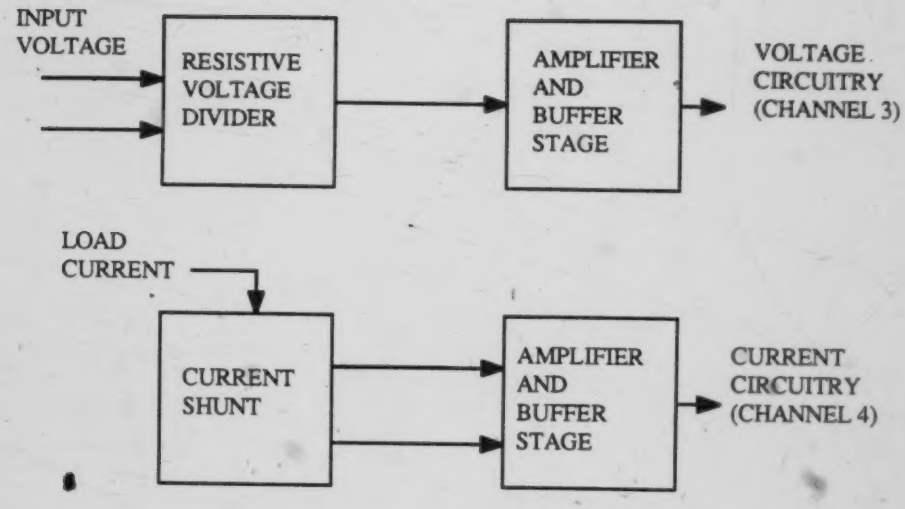


Fig. 3.14 Block diagram of the Phase 2 circuitry

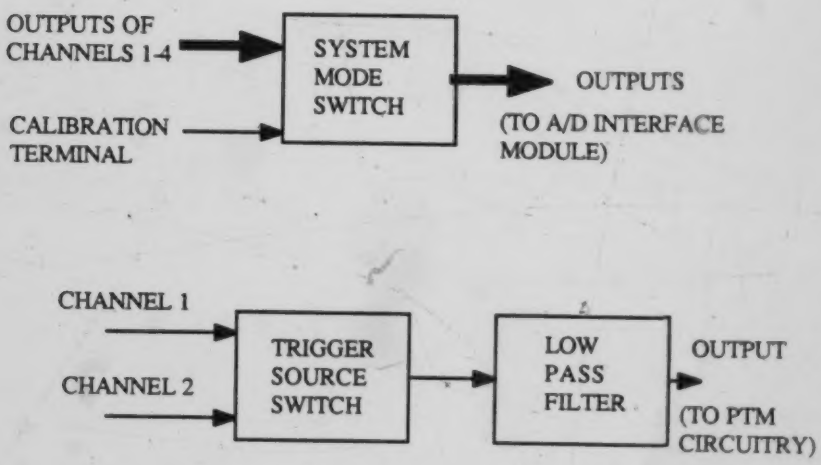


Fig. 3.15 Block diagram of the analog switches circuitry

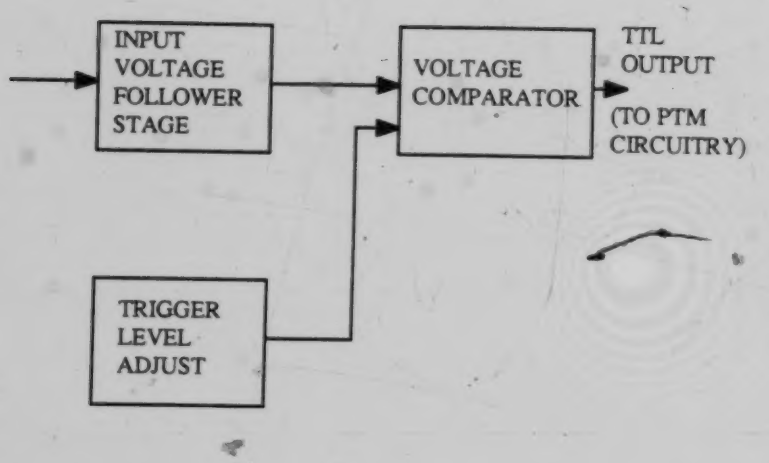


Fig. 3.16 Block diagram of the trigger circuitry

4. DIGITAL SAMPLING WATTMETER SOFTWARE

The software structure is geared towards the efficient and speedy implementation of the necessary mathematical manipulations and wattmeter functions required. This ties in suitably with the design goal of implementing a very accurate system using reasonably priced off-the-shelf components.

The hardware and software marriage is effectively achieved with 'idle' periods such as occur during sampling intervals being utilized for program code execution by the microprocessor.

With the ease of use and maintenance in mind, a user-friendly software interface is provided, as well as routines to simplify the system calibration and maintenance. The wattmeter program source code is included in Appendix B.

4.1 Software Overview

The flowchart in Fig. 4.1 (page 79) indicates the program flow of the digital sampling wattmeter.

On power-up or following a system reset, the system initialization and memory test are performed. This is followed by the execution of the main program.

4.2 Initialization and Memory test

The flow diagram of this section is shown in Fig. 4.2

(page 80). Initially, all system interrupts are disabled. Subsequently, the flip-flops and timing controller are initialized. This is followed by the initialization of the microprocessor peripheral devices namely; the 6821 PIA, the 8279 programmable keyboard and display controller, the 6840 PTM, the MC68488 GPIA ICs as well as the LCD display.

On successful completion of the above procedure, the memory test is performed. This test extensively checks all 64 kilobyte locations in the system static RAMS. A static ram test [3] with 5 data backgrounds is used. An indication of the current data background (0-4) being used is reflected on the corresponding LED of the bargraph display. A failure of this test results in the processor displaying the failing address . In the last-mentioned mode, a system reset is the only option available to the user in restarting the system.

The ensuing program code enables the keypad and performs the second initialization. In this initialization stage the voltage, current and power scale factors are stored in the static RAMS. Next, the program mode variables are initialized and all interrupts are enabled. Subsequently, the analog-to-digital interface module has its inputs switched to the outputs of channels 1 to 4. All microprocessor interrupts are enabled.

4.3 The main program

The flow diagram of this module is indicated in Fig. 4.3 (page 81). The program code interacts directly with the hardware in accepting user inputs as well as computing and displaying results as required through an interaction of system routines.

4.3.1 The main menus

The routines used in this stage are the input and LCD string display routines. The limitations of the LCD mentioned earlier prevent the display of the description of all the 32 possible keypad commands on the LCD display simultaneously. As a result the command summary is divided into 8 menus denoted menu 0 to menu 7. The screen display of these menus are indicated in Fig. 4.4 (page 82). The display of the next menu is always performed using the ^F command. The display of the previous menu being performed using the ^B command. A program command having a caret prefix indicates the simultaneous enabling of the control switch and the corresponding keypad character. Following the execution of a particular keypad command, the previous main menu displayed is always entered.

4.3.2 Command preprocessor

This section of the main program is always entered following the display of a particular main menu. The main

routine used is the input routine. The purpose of the command preprocessor is to determine the command entered and transfer program control to the required module. On entering a non-defined command, or in the absence of any command, the software simply loops through the command look-up table. The commands required by the program are passed through the byte-wide key buffer. The contents of the latter memory location is used by the command preprocessor software in transferring program control to the required module. The system hardware interacts with the key buffer either through the keypad or externally through the IEEE-488 bus (GPIB). Both sources operate in an interrupt mode.

The keypad can be operated in either the normal or the keypad programming mode. In the normal mode, commands are entered in response to the displayed menu messages. The command entered is duplicated in the key buffer by the interrupt service routine executed when a key closure occurs. The keypad programming mode is basically an emulation of the GPIB bus programming mode. In the GPIB bus programming mode, the user enters a programming code string through an active talker on the bus. This latter string, comprising the key code commands entered in the preferred order of execution, is stored in the 128 byte programming code buffer. The programming mode semaphore is also set in this mode. This enables the loading of the next program

byte from the program code string into the key buffer during the ensuing program execution.

The GPIB program commands are ; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U and V. The corresponding keypad commands are; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, ^0, ^1, ^2, ^3, ^4, ^5, ^6, ^7, ^8, ^9, ^A, ^B, ^C, ^D, ^E and ^F respectively.

4.3.3 Module 0

The flow diagram of this module is shown in Fig. 4.5 (page 83). This module is executed when a '0' is processed by the command preprocessor. This module restarts the wattmeter.

4.3.4 Module 1

The flow diagram of this module is shown in Fig. 4.6 (page 84). This module is executed when a '1' is processed by the command preprocessor. The routines used are the LCD and bargraph display routines. The function of this module is to toggle the display between the on and off states. The status of the display is reflected on LED 0 of the bargraph display using a bit in a byte-wide memory location. The bit manipulation instructions of the 68000 microprocessor enable the ease of implementation of the bargraph display.

4.3.5 Module 2

The flow diagram of this module is shown in Fig. 4.7 (page 85). This module is executed when a '2' is processed by the command preprocessor and uses the display and scale factor routines. The purpose of this module is to set the correct scale factors for the computation of the system parameters when either the 200v or the 350v range on channel 1 is being used. Following a system power-up, the 200v range is used. The range used is reflected on LED 3 of the bargraph display. Subsequently, the required voltage, current and power scale factors are computed using the scale factor routines. Following the latter procedure, the hex value FF is loaded into the key buffer. The hex value FF does not correspond to any key code. Hence, the transfer of program control to a particular module after a return to the command preprocessor module via the main menu, can only be effected when a new command is entered.

4.3.6 Module 3

The flow diagram of this module is shown in Fig. 4.8 (page 86). This module is executed when a '3' is processed by the command preprocessor. The function of this module is similar to that of module 2. However in this case the range indication refers to channel 3. In addition, the range status is indicated on LED 4.

4.3.7 Module 4

The flow diagram of this module is shown in Fig. 4.9 (page 87). This module is executed when a '4' is processed by the command preprocessor. This module uses the bargraph display routine and one of the utility routines namely; the overload test routine. The purpose of this module is to cause an execution of the overload routine or cause the program to skip over this routine following the execution of the acquisition routine. The software uses the state of the overload semaphore in determining the execution or otherwise of the overload routine. The program default mode following a system power-up or reset, is the execution of the overload test. This module is relevant during a system calibration or maintenance. The system status regarding the execution of the overload test is reflected on LED 1 of the bargraph display. Following the latter procedure, the hex value FF is loaded into the key buffer. The latter approach forces a return to the main menu and also prevents the re-execution of this module following the return.

4.3.8 Module 5

The flow diagram of this module is shown in Fig. 4.10 (page 88). This module is executed when a '5' is processed by the command preprocessor and uses the LCD display routine. The primary use of this module is the changing of the numerator of the voltage or current scale factors. This module is relevant when new scale factors need to be

programmed as a result of the replacement or aging of components. Following the execution of a '5', the sub-menu in Fig. 4.11 (page 89) is displayed. After selecing the appropriate channel, the user selects 'F', 'B' or ^6 to increment, decrement or terminate scale factor updating respectively. Subsequently, the hex value FF is loaded into the key buffer.

4.3.9 Module 6

The flow diagram of this module is shown in Fig. 4.12 (page 88). This module is executed when a '6' is processed by the command preprocessor. This module uses the LCD display routine and is similar to module 5. However, the primary use of this module is the changing of the exponent of the denominator of the voltage or current scale factors. The denominators of each scale factor have a mantissa of 2.

4.3.10 Module 7

The flow diagram of this module is shown in Fig. 4.13 (page 90). This module is executed when a '7' is processed by the command preprocessor. This module uses the bargraph display routine. Under software control using the PIA, the trigger source switch is employed in selecting either channel 1 or channel 2 as the system trigger source. The trigger source status is indicated on LED 1 of the bargraph display. Following a system power-up or reset, channel 1 is

used as the system trigger source. The hex value FF is loaded into the key buffer prior to exiting this module.

4.3.11 Module 8

The flow diagram of this module is shown in Fig. 4.14 (page 91). This module is executed when a '8' is processed by the command preprocessor. This module reinitializes the data storage parameters. This module sets the size of the data stored in RAM to zero in addition to setting the data pointer to the base of the RAM location. The data pointer points to the next free memory location for the storage of the LCD display data. Following the execution of this module, a return to the main menu is performed.

4.3.12 Module 9

The flow diagram of this module is shown in Fig. 4.15 (page 92). This module is executed when a '9' is processed by the command preprocessor. This module uses the AC acquisition, display and AC computation routines. The AC acquisition routine enables the acquisition of samples for a single cycle of the input signal. The AC computation routines utilize the procedure mentioned in section 2.3.4. The purpose of this module is to compute and display the rms value of the sampled data in the presence of an AC signal on the calibration terminal. On entering this module, the system mode switch is switched over to the

calibration terminals under software control. The sub-menu of Fig. 4.11. is next displayed. The user next selects the required channel. This is followed by the initialization of the PTM as well as the timing controller. The acquisition routine is next executed. Prior to the execution of the acquisition routine, the keypad is disabled through the PIA and enabled following the completion of the data acquisition. The multiplication, summation and square-root computation routines are then employed in determining the root-mean-square value of the acquired data. The results are subsequently displayed on the LCD display unit. The displayed values are then stored in the storage location in ram provided the storage of data has previously been enabled. The above procedure is executed until another command is entered. Following that, the system mode switch is connected to channels 1 to 4 and program execution returns to the main menu.

4.3.13 Module 10

The flow diagram of this module is shown in Fig. 4.16 (page 93). This module is executed when an 'A' is processed by the command preprocessor. This module uses the DC acquisition, display and DC computation routines. The DC acquisition routine results in the acquisition of 1024 samples. The DC computation routines utilize the trapezoidal rule procedure mentioned in section 2.3.2. The

purpose of this module is to perform a further trimming of the digital value corresponding to a zero input. This is achieved by adding the average of the DC values obtained for 256 acquisition cycles to the default zero value. Following a system power-up, the default zero value is 800hex. On entering this module, the sub-menu of Fig. 4.11. is displayed. Thus the user can select the channel for recalibration. This module has the advantage of enabling the system to be periodically readjusted to its optimum state in software. This is easily performed in a programmable mode of system operation.

4.3.14 Module 11

The flow diagram of this module is shown in Fig. 4.17 (page 94). This module is executed when a 'B' is processed by the command preprocessor and uses the acquisition, display and computation routines. The purpose of this module is to compute and display the voltage, current , power and power factor depending on the mode selected. On entering this module, the following sub-menu is displayed on the LCD unit.

PRESS:

1 FOR PHASE 1

2 FOR PHASE 2

3 FOR 3-PHASE

Following the user's selection, the PTM and timing controller are initialized. The presence or absence of an AC signal is next determined by software. The presence of an AC signal results in the use of the AC mode routines. The absence of the latter signal results in the use of the DC routines. In the AC single-phase mode, following the acquisition of the samples, the voltage, current and power are computed using the required scale factors. The power factor is subsequently computed and all four parameters are displayed. In the 3-phase mode, the power for both phases is computed and displayed. In addition, the power factor assuming a 3-phase three-line balanced system, is computed and displayed. In the DC mode, following the acquisition of the samples, the voltage, current, power are computed using the required scale factors. The above procedure is re-executed till another command is entered.

4.3.15 Module 12

The flow diagram of this module is shown in Fig. 4.18 (page 95). This module is executed when a 'C' is processed by the command preprocessor and uses the acquisition, display and computation routines. This module uses a signal

on the calibration terminals. The average of the sampled values obtained during an acquisition cycle are computed and displayed in hexadecimal notation. This procedure serves as an aid to calibrating the device.

4.3.16 Module 13

The flow diagram of this module is shown in Fig. 4.19 (page 96). This module is executed when a 'D' is processed by the command preprocessor and uses the display and data entry mode routines. On entering this module, the user enters a 4-digit number acquisition cycle value. This mode of system operation works with the following keypad commands: 9,B,C and F. The acquisition cycle value is simply the number of acquisition cycles to be displayed in the required module. A zero acquisition cycle value corresponds to the continuous acquisition case. On power-up or following a system reset, the acquisition value is zero.

4.3.17 Module 14

The flow diagram of this module is shown in Fig. 4.20 (page 97). This module is executed when an 'E' is processed by the command preprocessor and uses the display and data entry mode routines. On entering this module, the user enters a 4-digit number delay value. This mode of system operation works with the following keypad commands: 9,B,C

and F.The delay value simply corresponds to the number of results computed between successive display updates. Once the meter is operational, the delay value can only be changed by executing this module. On power-up or following a system reset, the delay value is zero.

4.3.18 Module 15

The flow diagram of this module is shown in Fig. 4.21 (page 98). This module is executed when an 'F' is processed by the command preprocessor and uses the computation and display routines. The module computes and displays the frequency of the trigger signal. In addition, the maximum and minimum frequencies computed since entering this module are displayed. The system utilizes one of the timers of the 6840 PTM configured in the frequency measurement mode. A software polling scheme of the internal interrupt bit is employed in determining the counter value corresponding to one cycle of the trigger input signal. This results in a lower measurable frequency limit than can be obtained using a hardware interrupt scheme. The latter improvement is due to the higher interrupt latency.

4.3.19 Module 16

The flow diagram of this module is shown in Fig. 4.22 (page 99). This module is executed when the command '^0' is processed by the command preprocessor and uses the

display, the scale factor and data entry mode routines. This module enables the user to enter the current shunt value in use. On entering this module, the following sub-menu is displayed.

```
^0: ENTER SHUNT VALS
 1: FOR CHANNEL 1
 2: FOR CHANNEL 2
 0  TO EXIT
```

The user performs the required selection and enters the shunt value. As in all menus and sub-menus, a wrong selection is simply ignored.

4.3.20 Module 17

The flow diagram of this module is shown in Fig. 4.23 (page 100). This module is executed when the command '^1' is processed by the command preprocessor and uses the display routines. This module indicates the size of the stored data in terms of the number of lines they would occupy on the display. The result is displayed for approximately 2.5 seconds. Program execution is subsequently returned to the main menu.

4.3.21 Module 18

The flow diagram of this module is shown in Fig. 4.24 (page 101). This module is executed when the command '^2' is processed by the command preprocessor and uses the

display routines. This module displays the size of the stored data in terms of the number of lines they would occupy on the display for approximately 2.5 seconds. In addition, the actual stored data is also displayed for approximately 5 seconds. This procedure is executed until either a new command is entered or all the data is displayed. Program execution is subsequently returned to the main menu.

4.3.22 Module 19

The flow diagram of this module is shown in Fig. 4.25 (page 102). This module is executed when the command '^3' is processed by the command preprocessor and uses the bargraph display routine. This module enables the storage of the appropriate data and indicates this state on LED 5 of the bargraph display. Program execution is subsequently returned to the main menu.

4.3.23 Module 20

The flow diagram of this module is shown in Fig. 4.26 (page 102). This module is executed when the command '^4' is processed by the command preprocessor and uses the display and program entry routines. This module is the keypad programming mode referred to in Module 1. The user enters a program command string corresponding to the GPIB commands. On conclusion, the system automatically executes

the commands sequentially.

4.3.24 Module 21

The flow diagram of this module is shown in Fig. 4.27 (page 103). This module is executed when the command '^5' is processed by the command preprocessor. This command is one of the data entry mode routine commands. When this command is entered during data entry, the cursor is backspaced. This enables the user to correct any of the previously entered digits on the LCD display.

4.3.25 Module 22

The flow diagram of this module is shown in Fig. 4.28 (page 103). This module is executed when the command '^6' is processed by the command preprocessor. This command is one of the data entry mode routine commands. This command indicates to the data entry routine, the termination of data entry of digits on the LCD display.

4.3.26 Module 23

The flow diagram of this module is shown in Fig. 4.29 (page 104). This module is executed when the command '^7' is processed by the command preprocessor and uses the LCD display and data entry routines. The primary use of this module is the changing of the numerators and denominators of the voltage scale factors as well as the numerators of

the current scale factors. This module is relevant when new scale factors need to be programmed as a result of the replacement or aging of components. In this module the user is prompted to enter four digit numbers for voltage scale factors followed by two 4-digit numbers for the current scale factors. Program execution then returns to the main menu.

4.3.27 Module 24

The flow diagram of this module is shown in Fig. 4.30 (page 105). This module is executed when the command '^8' is processed by the command preprocessor. This module is basically an extension of module 10. The procedure executed by module 10, is performed automatically for all the channels. Thus the sub-menu of Fig. 4.11. is not displayed.

4.3.28 Module 25

The flow diagram of this module is shown in Fig. 4.31 (page 105). This module is executed when the command '^B' is processed by the command preprocessor and uses the input and display routines. The command results in the display of the penultimate menu.

4.3.29 Module 26

The flow diagram of this module is shown in Fig. 4.32 (page 106). This module is executed when the command '^C' is processed by the command preprocessor and uses the input and display routines. The command results in the display of the previous menu.

4.3.30 Module 27

The flow diagram of this module is shown in Fig. 4.33 (page 106). This module is executed when the command '^D' is processed by the command preprocessor. This command is one of the program entry mode routine commands. This command serves a dual function. It indicates the termination of program entry in the keypad programming mode as well as marking the end of the program string in both the keypad and GPIB programming modes.

4.3.31 Module 28

The flow diagram of this module is shown in Fig. 4.34 (page 107). This module is executed when the command '^E' is processed by the command preprocessor. This command is one of the program entry mode routine commands. When this command is entered during program entry, the cursor is backspaced. This enables the user to correct any of the previously entered program command string characters.

4.3.32 Module 29

The flow diagram of this module is shown in Fig. 4.35 (page 107). This module is executed when the command '^F' is processed by the command preprocessor and uses the input and display routines. The command simply results in the display of the next menu.

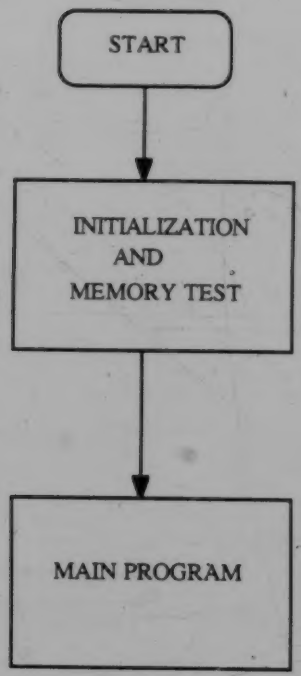


Fig. 4.1 Flowchart of the system software

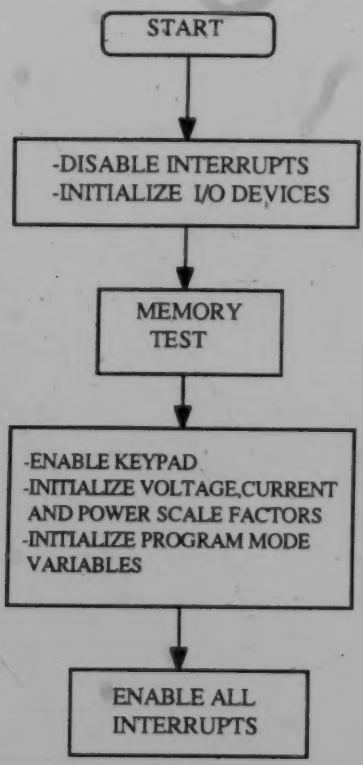


Fig. 4.2 Flow chart of the initialization and memory test

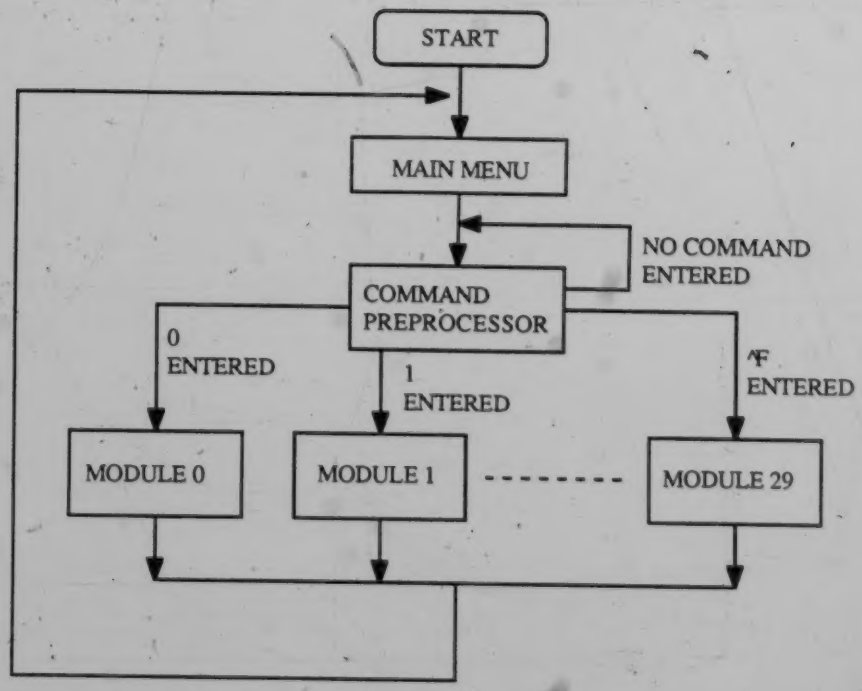


Fig. 4.3 Flowchart of the main program

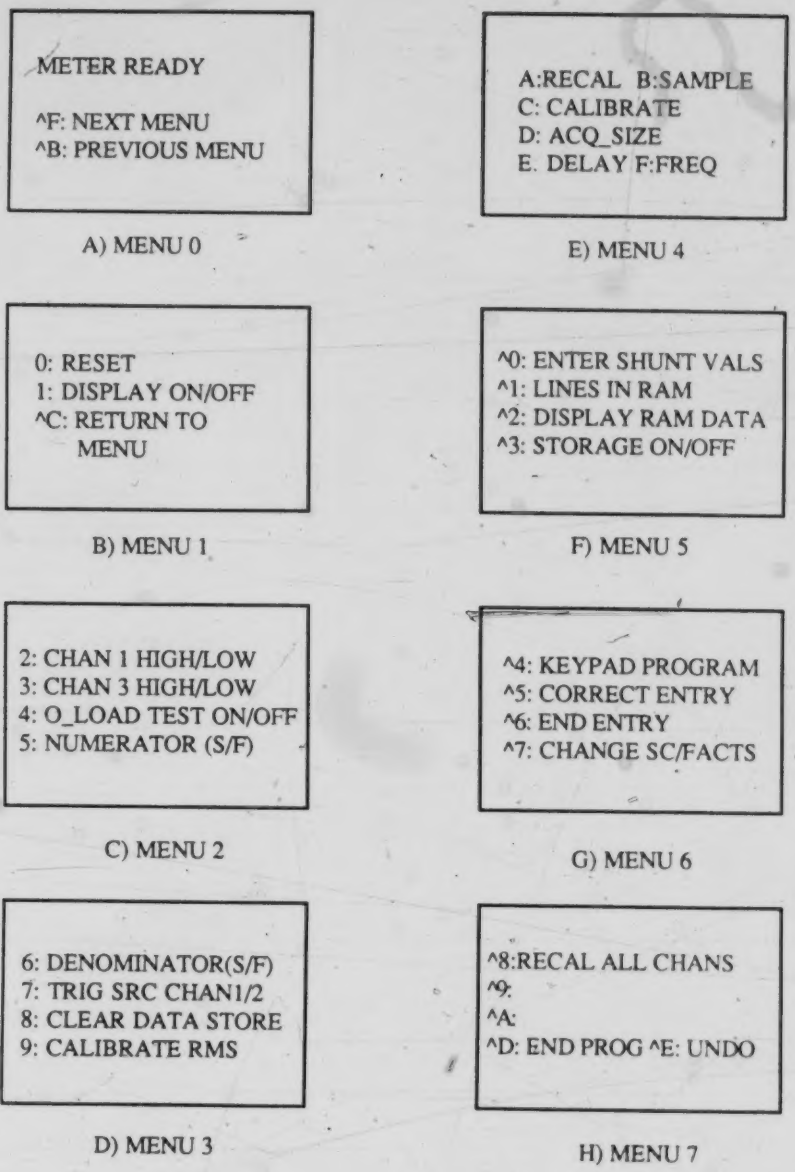


Fig. 4.4 Detailed main menu displays

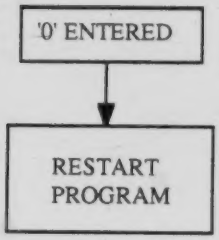


Fig. 4.5 Flowchart of module 0

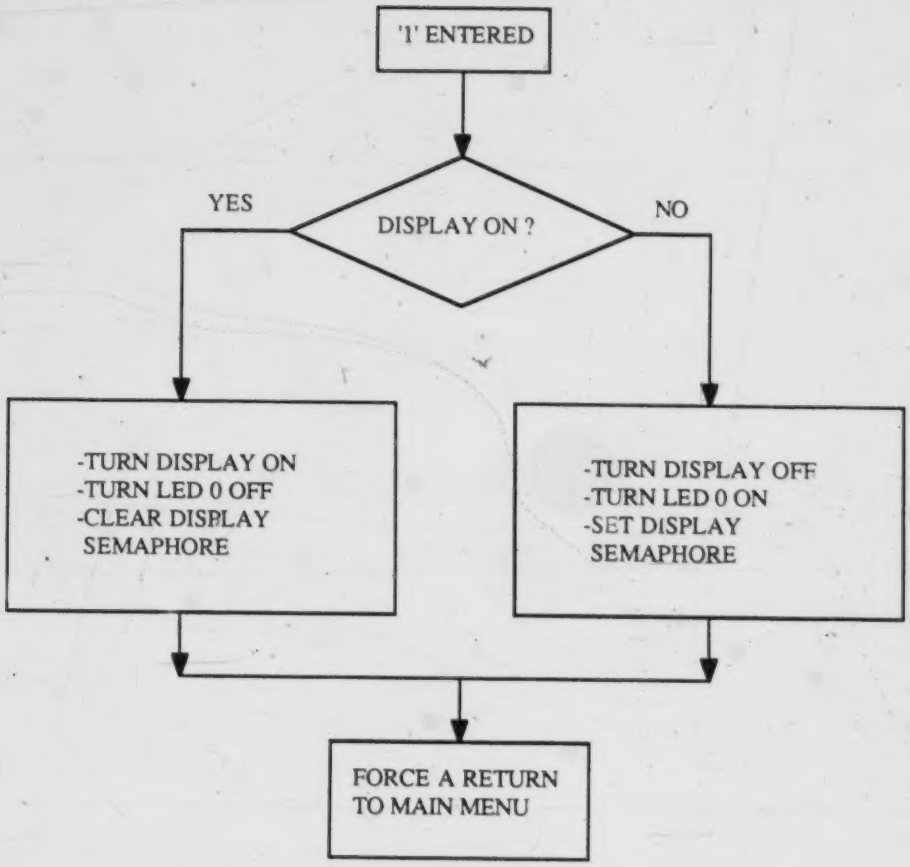


Fig. 4.6 Flowchart of module 1

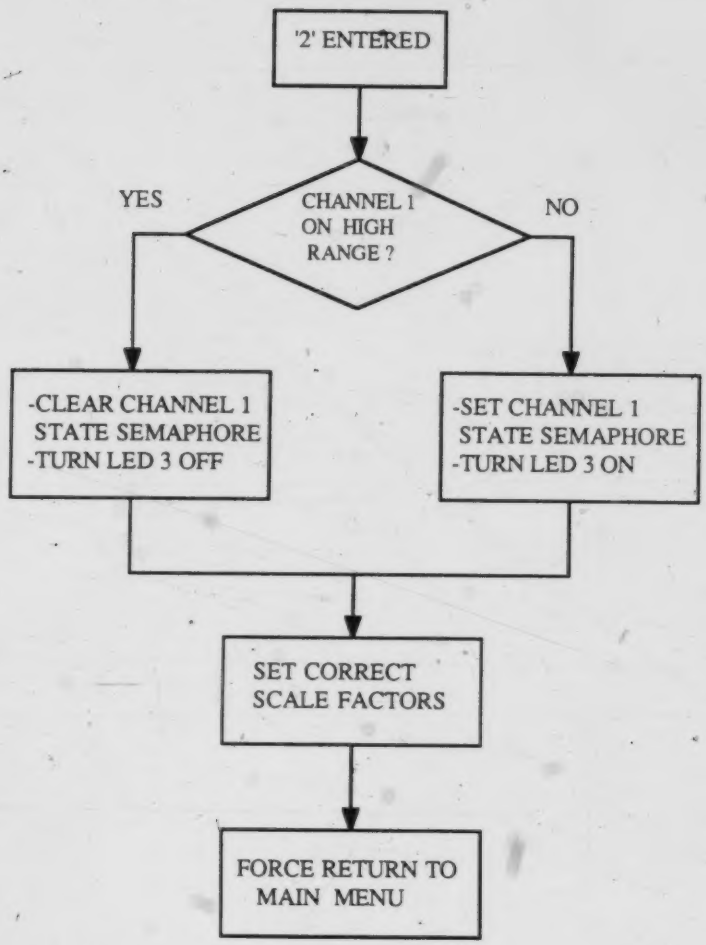


Fig. 4.7 Flowchart of module 2

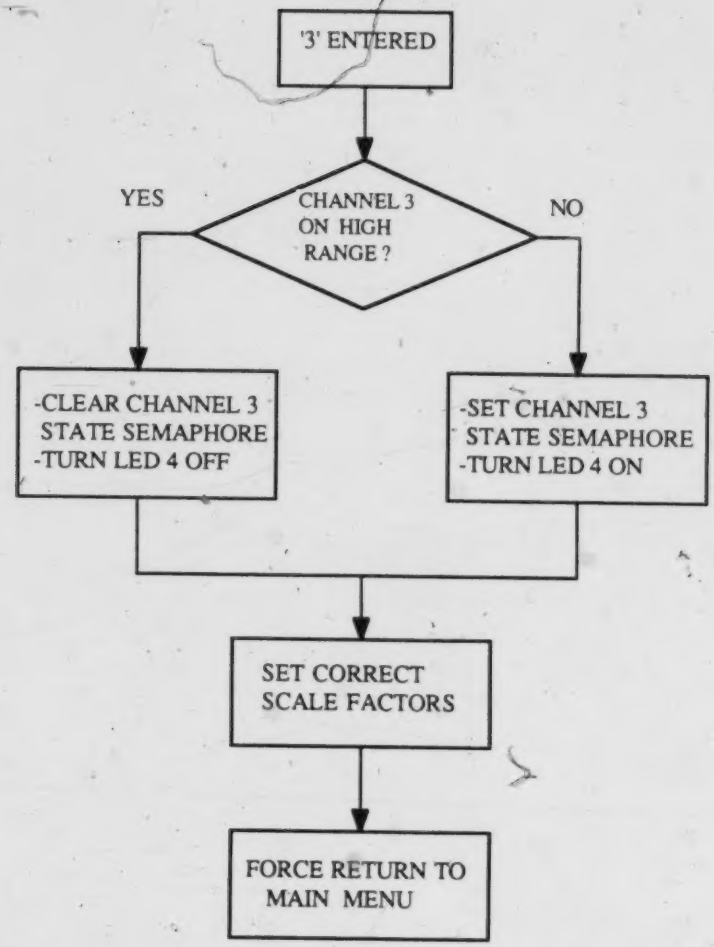


Fig. 4.8 Flowchart of module 3

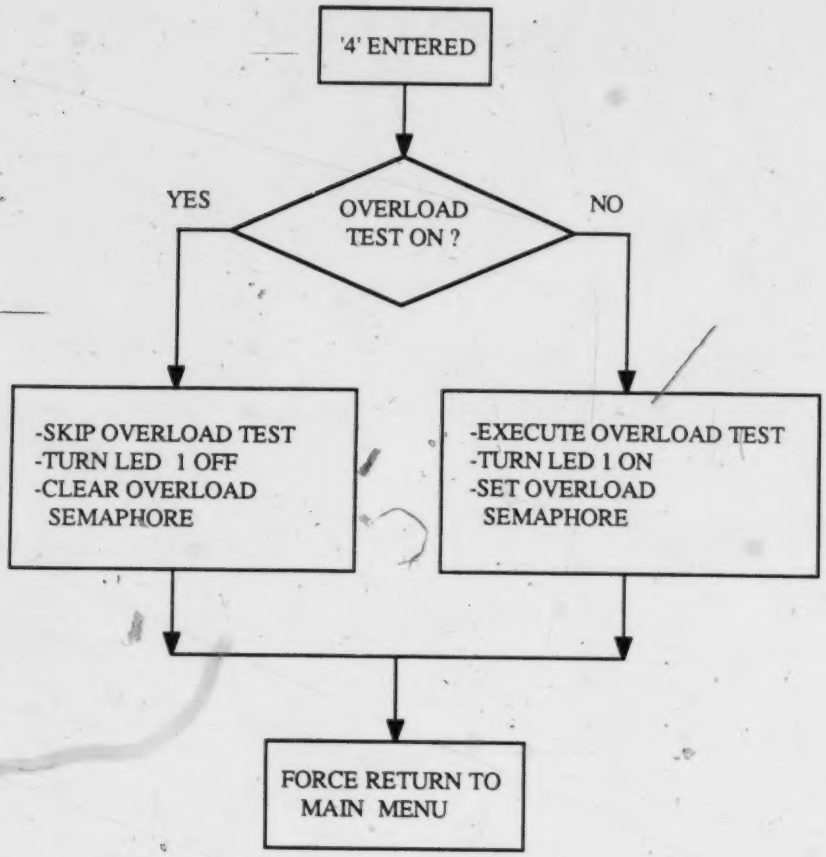


Fig. 4.9 Flowchart of module 4

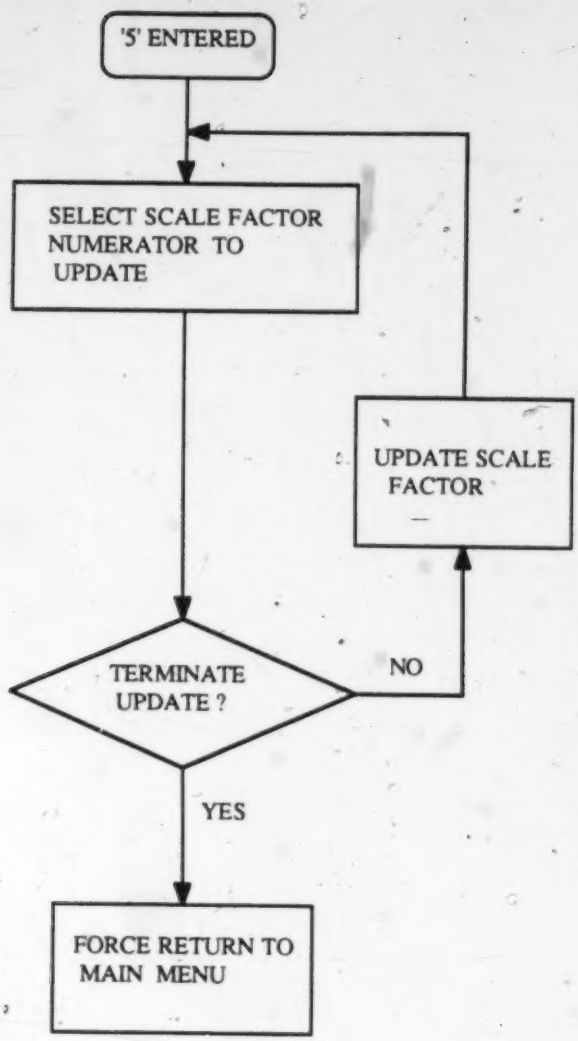


Fig. 4.10 Flowchart of module 5

1 FOR CHANNEL 1
2 FOR CHANNEL 2
3 FOR CHANNEL 3
4 FOR CHANNEL 4

Fig. 4.11 Channel selection sub-menu

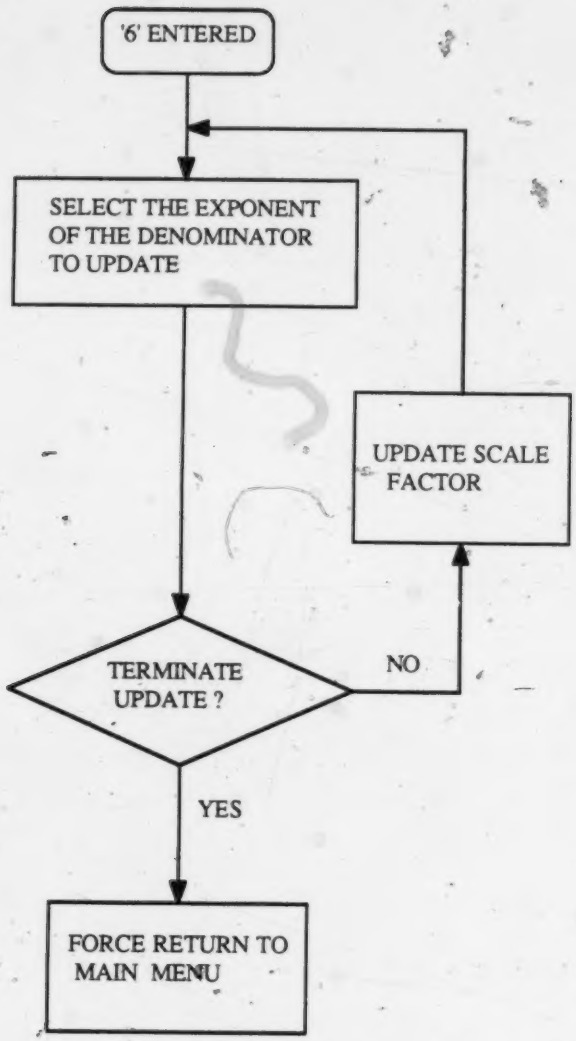


Fig. 4.12 Flowchart of module 6

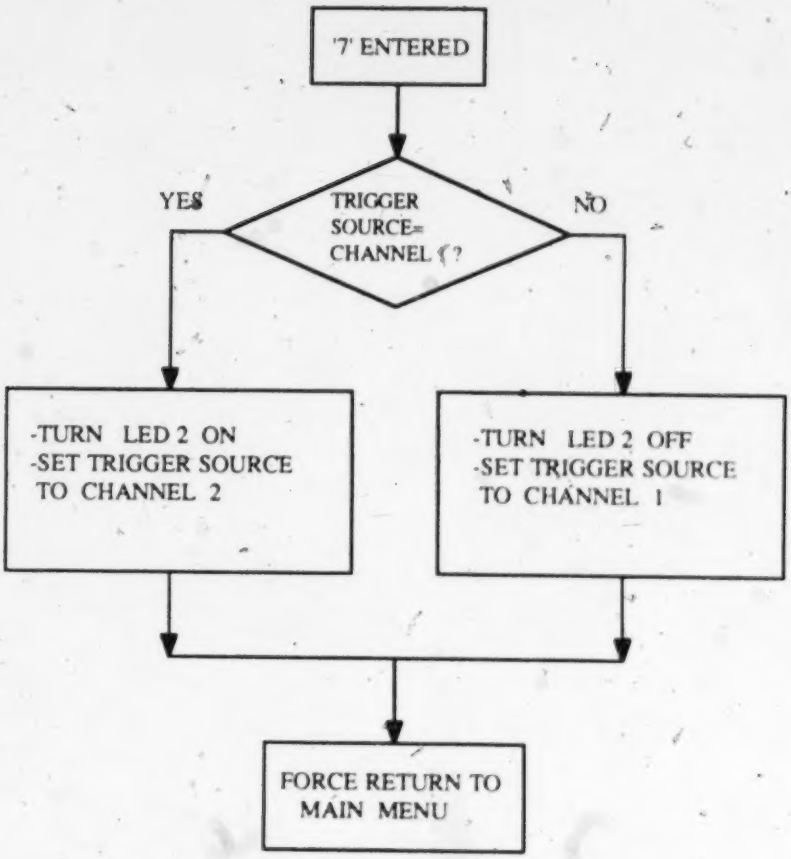


Fig. 4.13 Flowchart of module 7

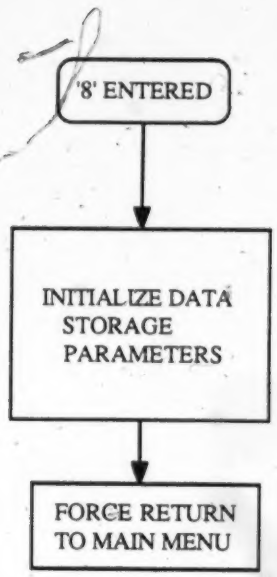


Fig. 4.14 Flowchart of module 8

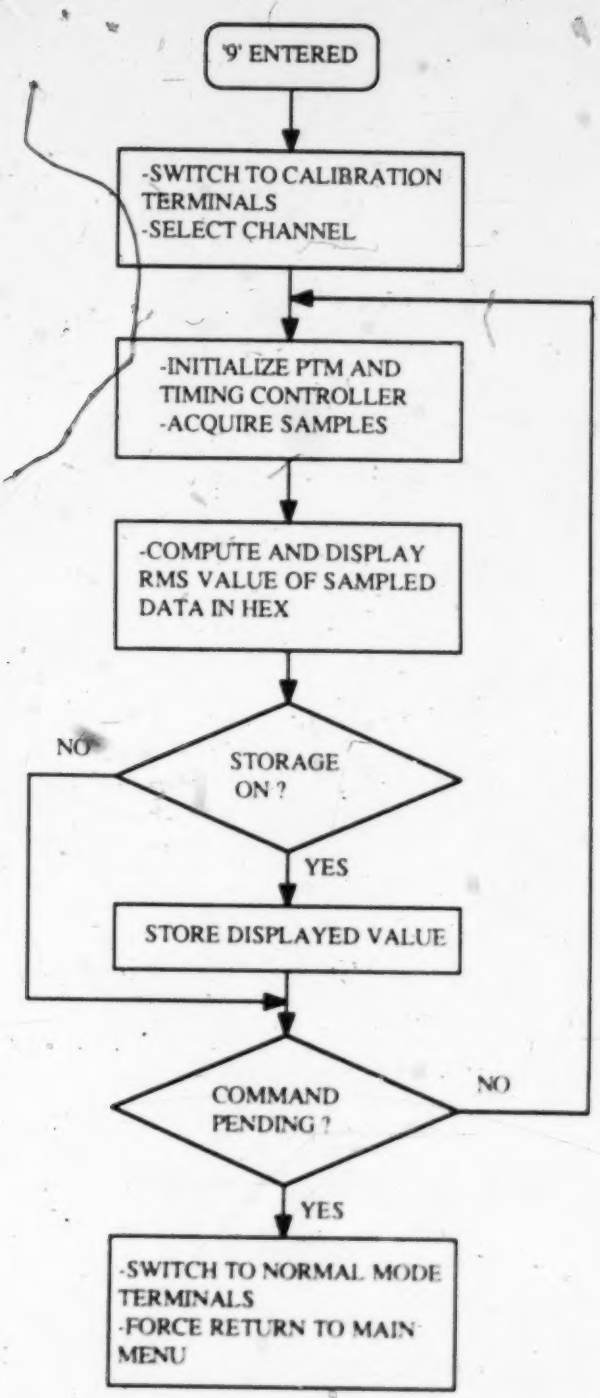


Fig. 4.15 Flowchart of module 9

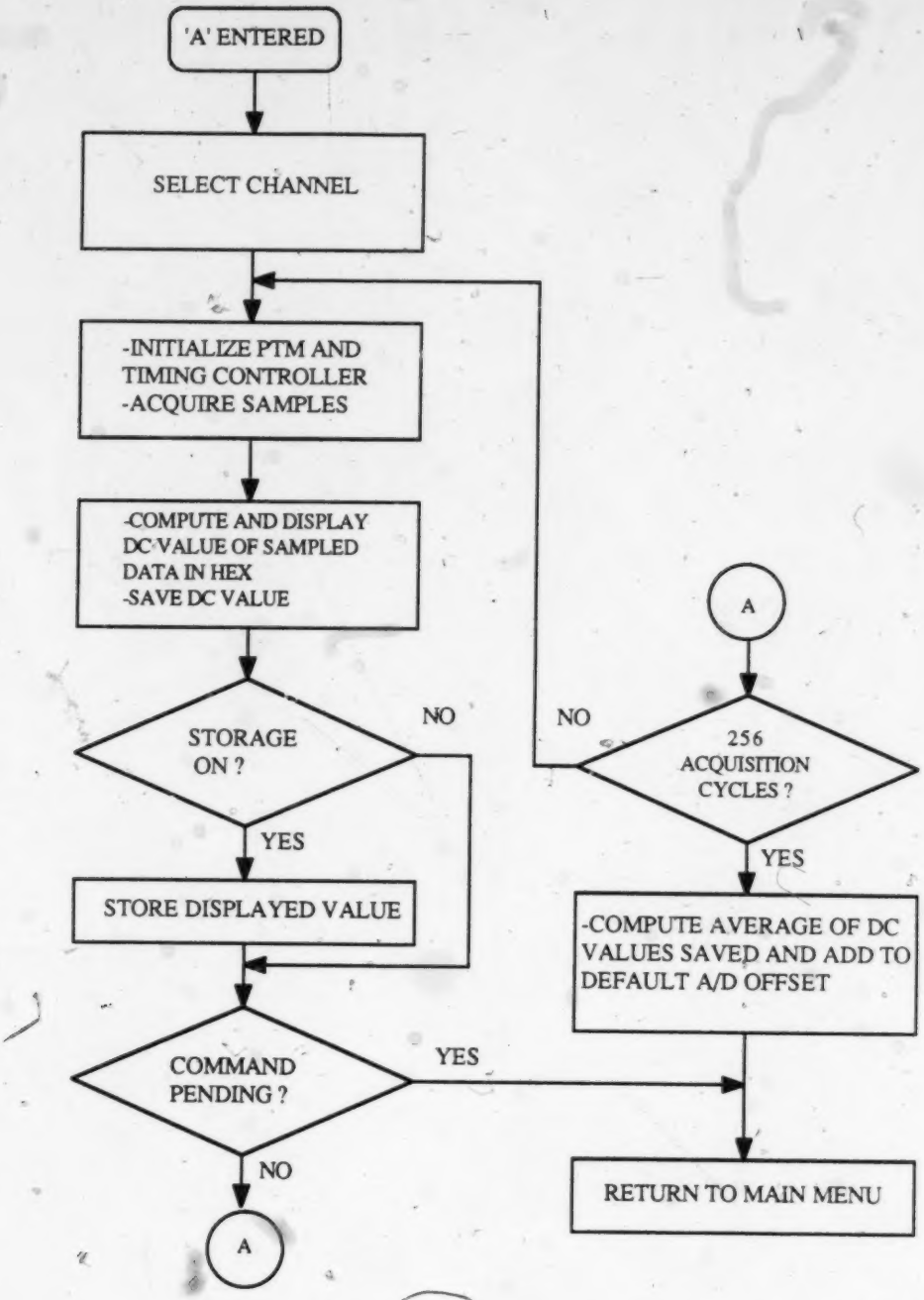


Fig. 4.16 Flowchart of module 10

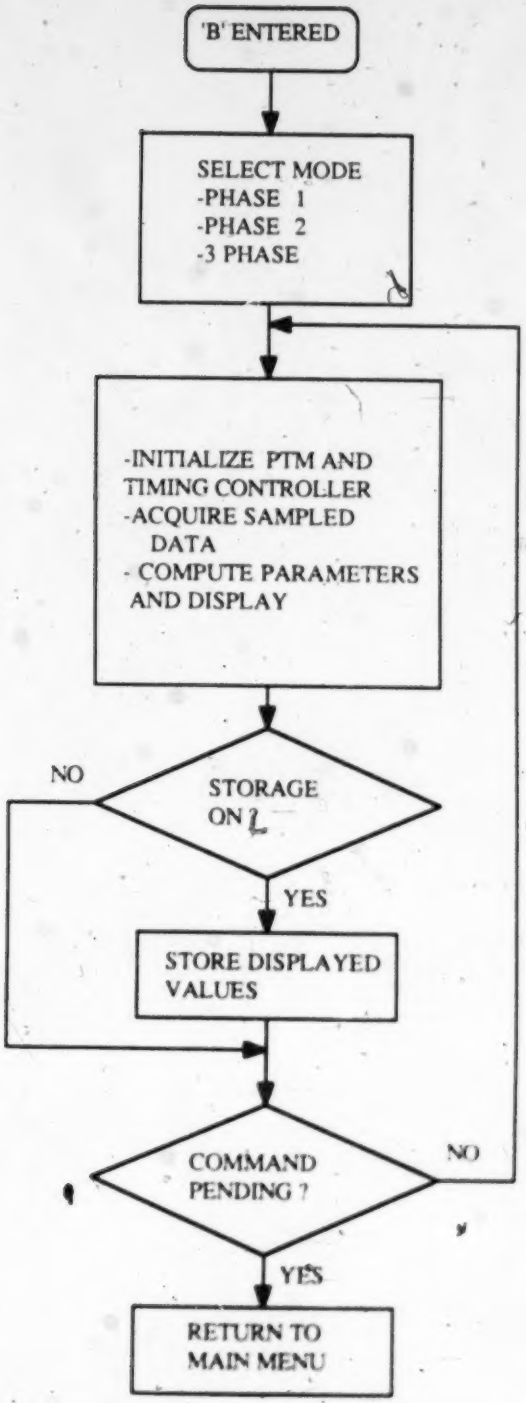


Fig. 4.17 Flowchart of module 11

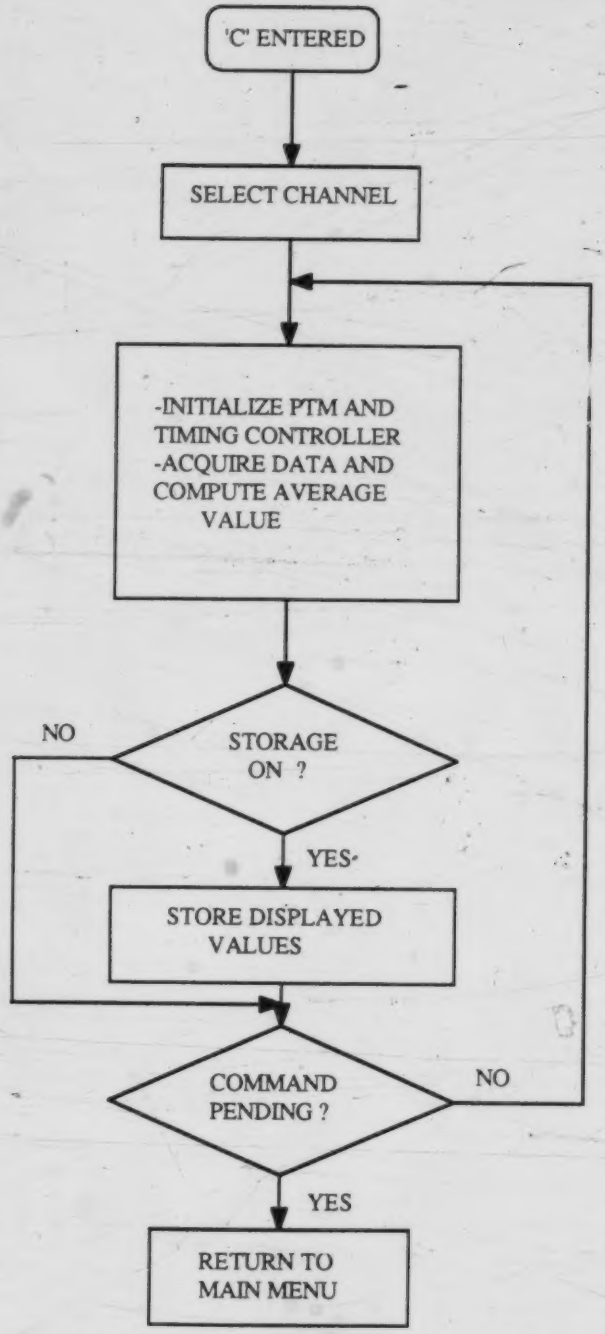


Fig. 4.18 Flowchart of module 12

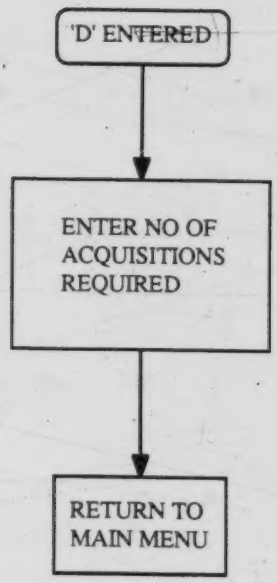


Fig. 4.19 Flowchart of module 13

Fig. 4.20 Flowchart of module 14

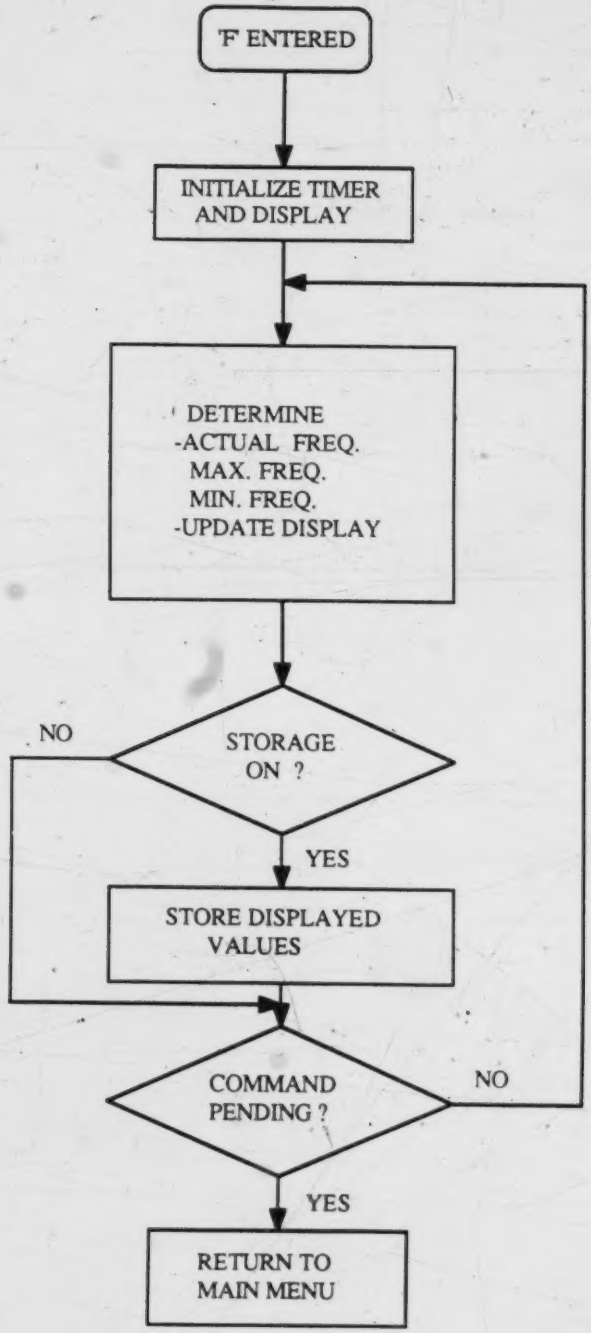


Fig. 4.21 Flowchart of module 15

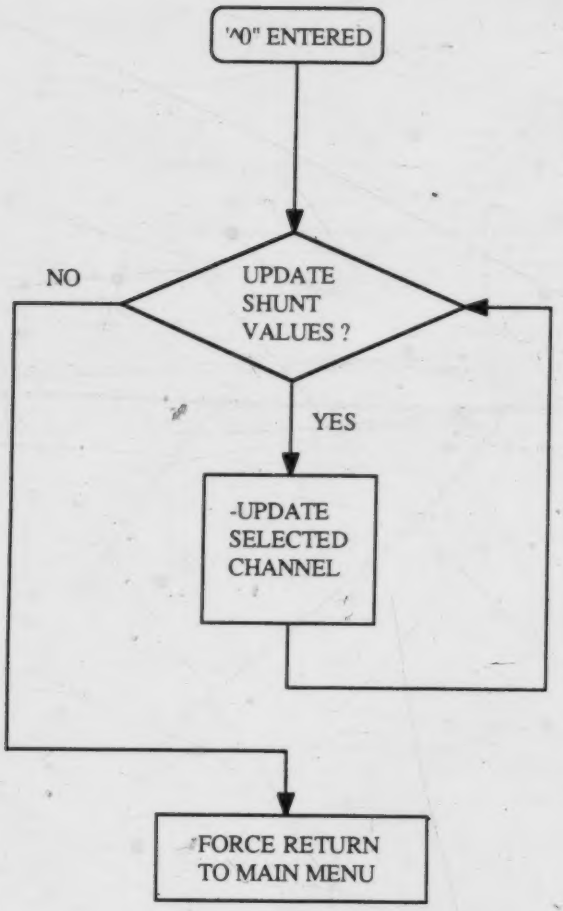


Fig. 4.22 Flowchart of module 16

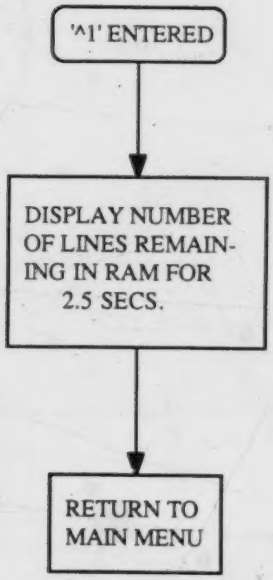


Fig. 4.23 Flowchart of module 17

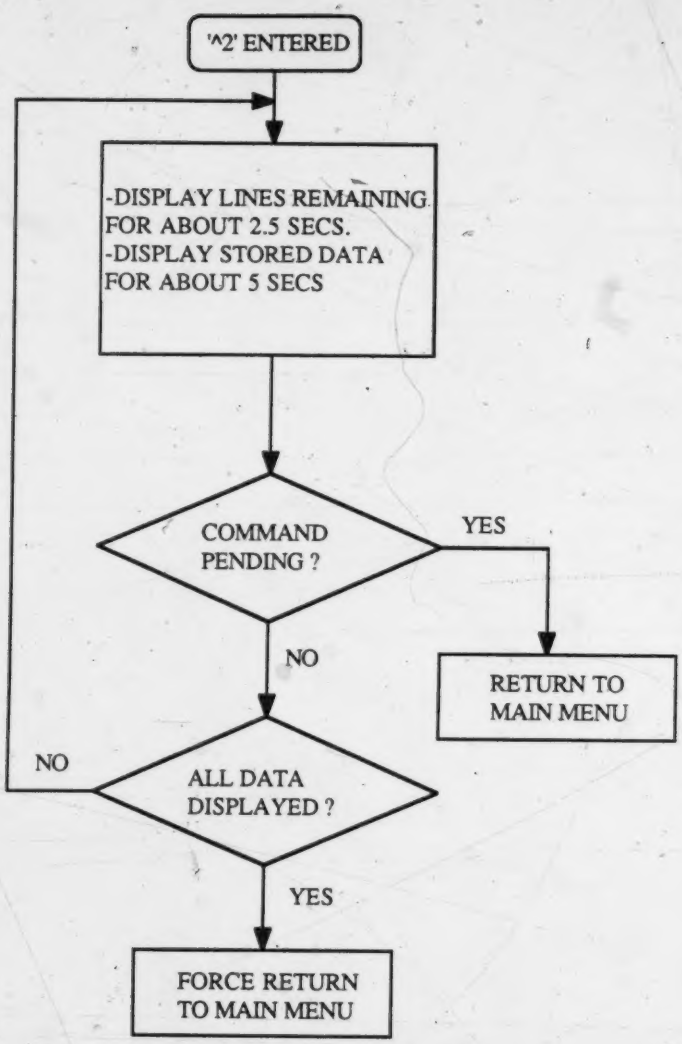


Fig. 4.24 Flowchart of module 18

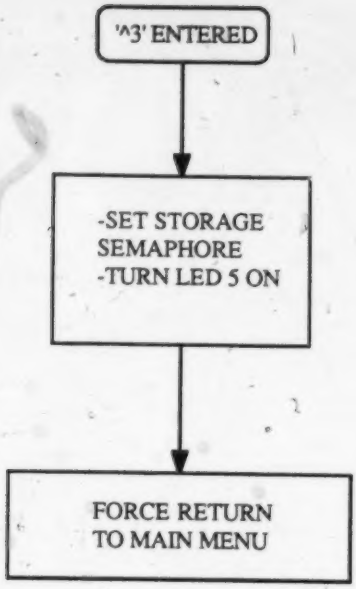


Fig. 4.25 Flowchart of module 19

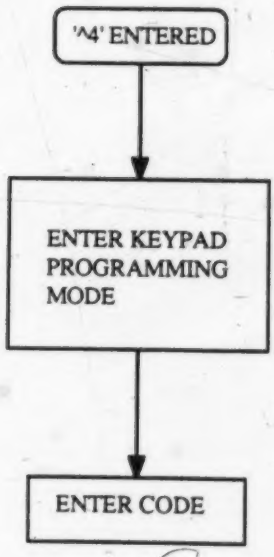


Fig. 4.26 Flowchart of module 20

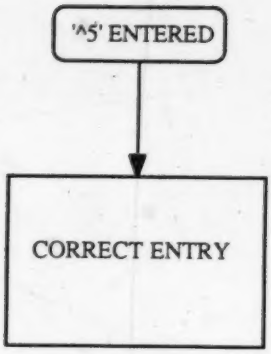


Fig. 4.27 Flowchart of module 21

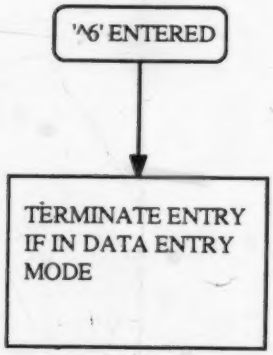


Fig. 4.28 Flowchart of module 22

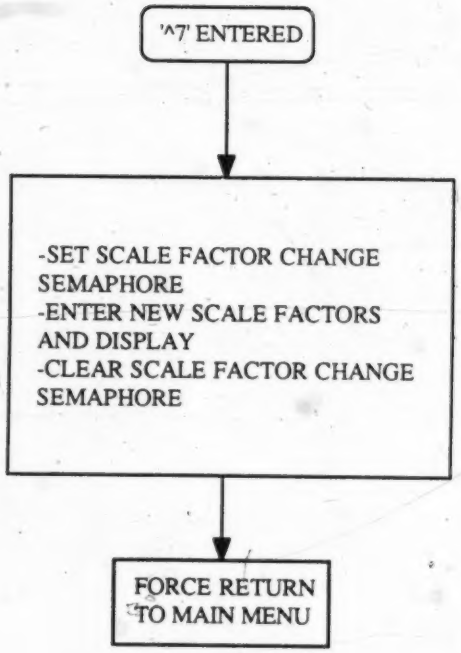

'^7' ENTERED
-SET SCALE FACTOR CHANGE SEMAPHORE
-ENTER NEW SCALE FACTORS AND DISPLAY
-CLEAR SCALE FACTOR CHANGE SEMAPHORE
FORCE RETURN TO MAIN MENU

Fig. 4.29 Flowchart of module 23

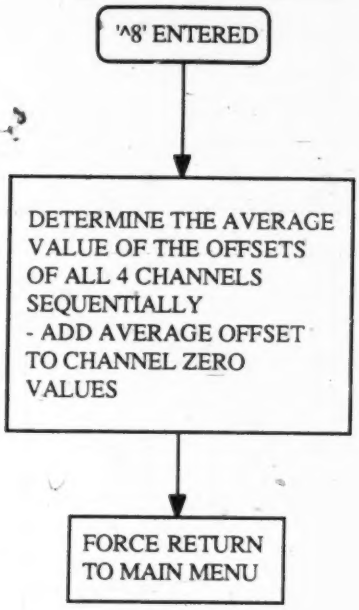

'^8' ENTERED
DETERMINE THE AVERAGE VALUE OF THE OFFSETS OF ALL 4 CHANNELS SEQUENTIALLY
- ADD AVERAGE OFFSET TO CHANNEL ZERO VALUES
FORCE RETURN TO MAIN MENU

Fig. 4.30 Flowchart of module 24

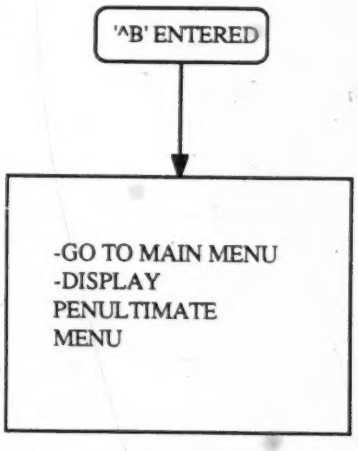


Fig. 4.31 Flowchart of module 25

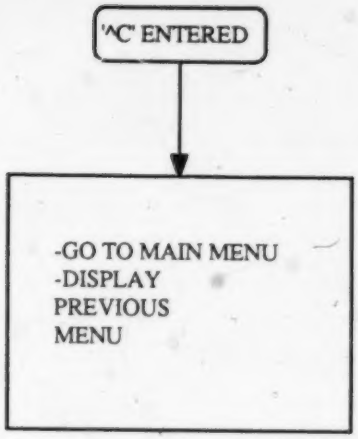


Fig. 4.32 Flowchart of module 26

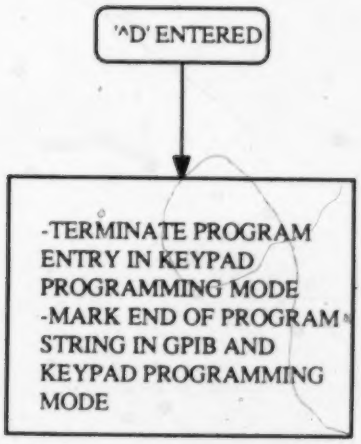


Fig. 4.33 Flowchart of module 27

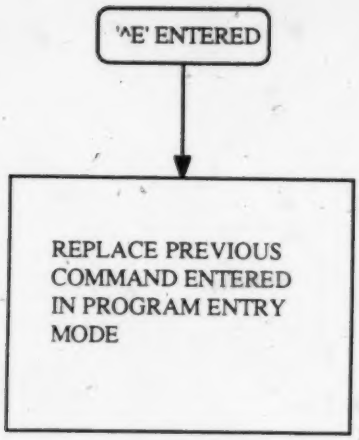


Fig. 4.34 Flowchart of module 28

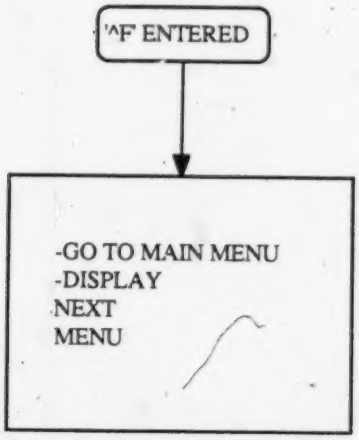


Fig. 4.35 Flowchart of module 29

5. SYSTEM ROUTINES

The program code utilizes various routines in the performance of it's tasks. These can be classified as; acquisition routines, computational routines and utility routines.

5.1 Acquisition routines

The main purpose of these routines is in transferring the digital outputs of the A/D converters to the storage media namely; the static ram ICs. During the transfer of the data, interrupts from the keypad are disabled and only re-enabled after the completion of the data transfer.

Detection of an AC or DC signal is performed using a software polling scheme of the relevant hardware lines.

In both AC and DC acquisition modes, following the issuance of a start of conversion signal, the microprocessor acquires the latched data. For each acquired 16-bit value, the respective channel 16-bit zero offset value is subtracted. The resulting two's complement value is subsequently stored in the static ram devices. All the above mentioned procedures are performed and completed well before the start of the next sampling interval. 1024 samples per channel are acquired in the DC mode. In addition to the above functions the AC mode acquisition routines determine the state of the sampling process during an AC acquisition cycle. This determination is performed

immediately after the results of the previously converted analog input signals of all 4 A/D converters have been stored in the system RAM. An AC acquisition cycle is the time interval during which the crossing signal (ZC) is at a logic high state as shown in Fig. 5.1 (page 117). Fig. 5.2 (page 117) shows the relationship between the crossing signal and the software tests for the end of an AC acquisition cycle. The above-mentioned software test also provides information regarding the state of the AC data acquisition process as obtained from the logic state of lines BD0 - BD4 of IC U33 in Fig. A.15. Based on the information obtained, a look-up table is used to determine the next stage of program execution as indicated below.

	Logic state of the relevant lines of IC U33				Action taken by routine
	BD3 (AC)	BD2 (2Q)	BD1 (ZC)	BD0	
a)	0	0	0	*	Start sampling again
b)	0	0	1	*	Start sampling again
c)	0	1	0	*	Start sampling again
d)	0	1	1	*	Start sampling again
e)	1	0	0	*	Stage 1 (An additional sample required)
f)	1	0	1	*	Continue (1st conversion underway)

g) 1 1 0 * Stage 2 (2 more samples required)

h) 1 1 1 * Continue (ZC high)

* - A don't care condition

Restarting the sampling procedure involves reinitializing the state controller flip-flops and counters as well as the PTM.

5.2 Computational routines

These routines are used in the determination of the various system parameters. The routines include floating point and fixed point routines. A C programming language floating point routines for a 68000 microprocessor target machine are used. In addition a conversion routine for a 64-bit fixed point to floating point notation and vice-versa is employed. Most of the computations involve fixed point routines resulting in a better computational time as opposed to the use of floating point routines. The major mathematical computations required in determining the parameters are those of multiplication, summation and square-root extraction. To speed up the computations extensive use is made of the microprocessor registers. In the scale factor routines, since the denominators consist of a mantissa of 2 and an exponent, determining the appropriate parameter values in their SI units, simply involves multiplications and shift operations.

The sign of the phase angle for the selected phase is

determined in software. This routine simply determines the sample number of the voltage and current samples which result in the respective signal maximum value. A higher sample number for the current signal indicates a lagging (-) power factor. The converse indicates a leading power factor.

5.3 Utility routines

These routines include the input and display subroutines, system state indication routines and a number of house-keeping routines.

The input routine comprises the keypad input routine and the GPIB input routine. The self-explanatory flow chart for the former routine is shown in Fig. 5.3 (page 118). This routine interacts with the keypad on an interrupt basis. The GPIB routine employs the GPIA device operating in an interrupt mode. This routine which employs interrupt service routine int_gpia is described below.

```
int_gpia:
    save all registers
    do {
    read character;
    switch;
    case 1 ( invalid character);
         break;
    Case 2( character = U )
```

```
            if character previously entered
            { decrement program counter;
            decrement program pointer; }
            break;
        Case 3 ;
            convert character to keypad code;
            store code in program buffer;
            break;
        }while ( character != 'T' or program counter < 128 )
        enddo
        restore all registers
    end int_gpia
```

The main display routines are the LCD and the bargraph display routines. There are two main LCD display routines; an ASCII character display routine and an ASCII string display routine. The former routine simply uses the LCD display unit instruction set and internal ASCII font table. The flow diagram of the string display routine is shown in Fig. 5.4 (page 119). The latter routine, requires an input ASCII string pointed to by register A5. The string should terminate with a zero. In addition, the line number of the display where the result should appear is passed to the routine through register D3 of the microprocessor. This routine preserves all the microprocessor registers.

The bargraph display routine the corresponding LED status correspond to the bits in 2 memory locations. LEDS

0 to 7 correspond to one memory location and LED 8 corresponds to a bit in the second memory location. Any required change in an LED status, is thus performed by initially changing the corresponding memory location bit, and finally writing the contents to the corresponding PIA port. The state of the bargraph display LEDS are summarized below.

LED number	Status when on
0	Display off
1	Overload Test off
2	Trigger Source = Channel 2
3	350v range selected on channel 1
4	350v range selected on channel 3
5	Storage of displayed data enabled
6	System in program mode
7	Storage space full
8	Keypad in remote mode

The data entry mode routine employs the input and LCD display routines. This routine is employed in the entry of numerals. This routine is centered around sub-routine acq. The latter subroutine which preserves all registers is described below.

```
acq:
    Save working registers;
    Clear display and digit counter;
    Display module message;
```

```
do {
Read keycode;
Switch;
Case 1( Invalid keycode)
          break;
Case 2( keycode corresponds to ^5 )
          if numeral previously entered
          { move display cursor back one space };
          break;
Case 3:   -Store keycode in program buffer
          -Display numeral entered
          -Update digit counter
)while ( keycode != ^6 and digit counter != 4 ).
enddo
restore working registers;
return from subroutine;
end acq:
```

The program entry mode routines comprise the keypad programming routines and the GPIB routines that have already been described. The first-mentioned routine uses the input and LCD display routines. This routine is centered around subroutine automode. The latter subroutine which preserves all registers, is described below.

```
automode:
    Save working registers;
    Initialize program mode variables;
```

```
Clear display;
do {
Read keycode;
Switch;
Case 1( Invalid keycode)
          break;
Case 2( keycode corresponds to ^D )
          if character previously entered
          { decrement program counter;
          decrement program pointer; }
          break;
Case 3:   -Store keycode in program buffer
          -Display corresponding GPIB command
          -Update program size counter
}while ( keycode != ^E or program size < 128 bytes )
enddo
restore working registers;
force return to main routine;
end auto_mode:
```

The system state indication routines are simply routines geared towards the ease of maintenance of the system. These routines utilize the microprocessor exception vectors excluding those allocated to the system device interrupts. Following an exception, the address causing the exception as well as the type of exception is displayed. In

this mode only a system reset can restart the system.

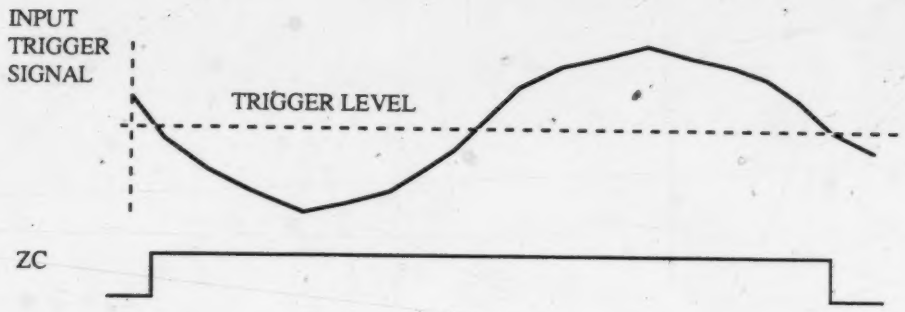


Fig. 5.1 Crossing detector timing diagram 1

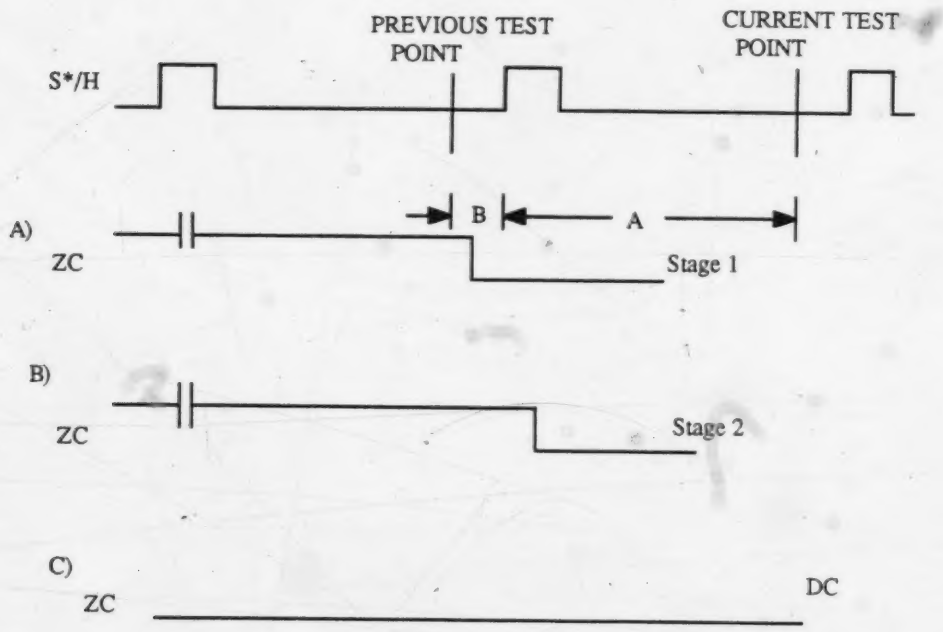


Fig. 5.2 Crossing detector timing diagram 2

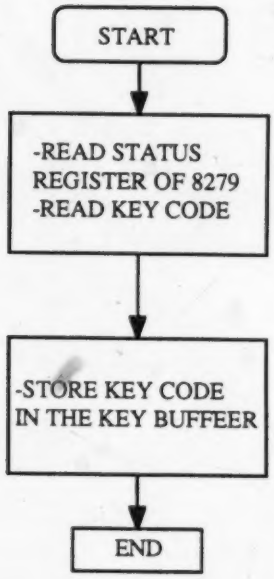


Fig. 5.3 Flowchart of the input routine

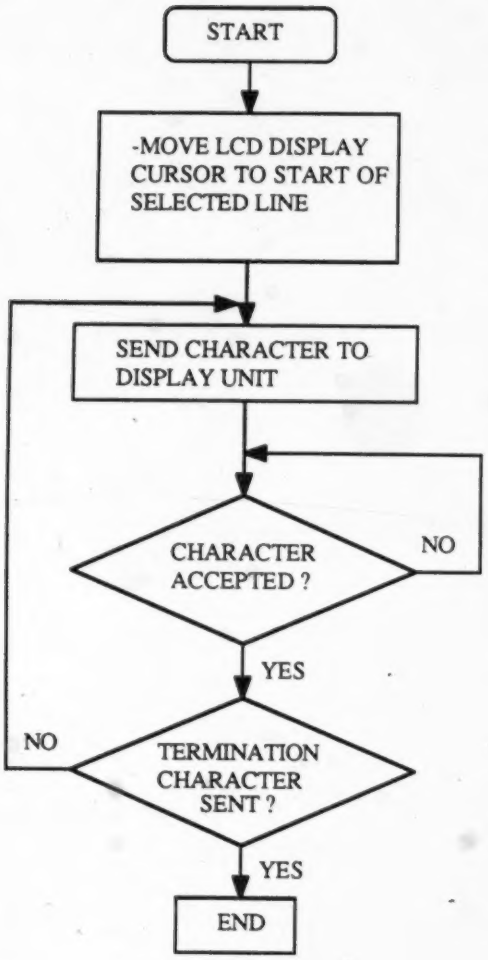


Fig. 5.4 Flowchart of the LCD display routine

6. SYSTEM OPERATION, PERFORMANCE AND SPECIFICATIONS

The system commands and their effects on the wattmeter operations have already been described. A convenient order of operation is however detailed. Following the power-up of the system, when the first menu is displayed, the shunt values can be entered. This can be followed by entering the appropriate voltage ranges. The command B is next entered and the variable resistors VR2, VR5, VR8 and VR11 are adjusted until the displayed parameters V1, I1, V2 and I2 are zero. Following this, the appropriate input terminals can be connected to the signal sources. The required measured value can be obtained by selecting the corresponding command.

6.1 Calibration

It is recommended that the system be up and running for at least a couple of hours prior to undertaking this procedure. This ensures that the system would have settled down to it's quiescent operating state. A standard voltmeter and ammeter, a variable voltage source 0 - 250V ac and a variable alternating current source are required for this procedure.

The command '4' is entered to disable the overload test. The command 'B' is entered, thus placing the system in the normal mode. The respective inputs of each channel are connected together. The corresponding offset adjustment

resistors VR14, VR15, VR17 and VR18 are adjusted until a zero volt reading is recorded on the respective outputs CH1-CH4 of the analog switches.

Calibration of channels 1 and 3 are performed as follows. The system is set to the calibration mode by entering the command 'C'. Channel 1 is selected and the calibration terminal grounded. Variable resistors VR2 and VR4 indicated in Fig. A.17 are adjusted until a reading of 000800 is obtained. Next, the system is switched to channel 3. Variable resistors VR8 and VR10 indicated in Fig. A.20 are adjusted until a reading of 000800 is obtained. The command 'B' is next entered and various input voltages applied to the respective voltage inputs using the variable voltage source. The corresponding gain setting variable resistors VR3 and VR9 are adjusted until the meter reading corresponds to that of the standard voltmeter.

The system is set to the calibration mode by entering the command 'C'. Channel 2 is selected and with the calibration terminal still grounded, variable resistors VR5 and VR7 indicated in Fig. A.18 are adjusted until a reading of 000800 is obtained. Next, the system is switched to channel 4. Variable resistors VR11 and VR13 indicated in Fig. A.21 are adjusted until a reading of 000800 is obtained. The command 'B' is next entered and various input currents applied to the respective shunts using the variable current source. The corresponding gain setting

variable resistors VR6 and VR12 are adjusted until the meter reading corresponds to the reading of the standard ammeter.

After calibrating all channels, the command '4' is entered to enable the overload test.

6.2 Scale factor calculations

The scale factors required for each channel where calculated by determining the ratio between the corresponding A/D input and the input voltage for various input voltages and a fixed load. A fine adjustment of the values obtained was performed in software to produce the following values.

Channel 1 (low range) - 2020/2^11 = 0.986

Channel 1 (high range) - 3638/2^11 = 1.776

Channel 2 (10 A shunt) - 201/2^10 = 0.196

Channel 3 (low range) - 2020/2^11 = 0.986

Channel 3 (high range) - 3638/2^11 = 1.776

Channel 2 (10 A shunt) - 201/2^10 = 0.196

The current and voltage scale factors are employed in obtaining the required power scale factors. The voltage scale factors when multiplied by the 2's complement of the corresponding A/D outputs result in a voltage reading 10 times the actual value. A shift in the decimal point one place to the left thus produces the required value to one decimal place. The voltage ratios for the low range

indicate that a 1 LSB change in the channel 1 or 3 A/D converter, results in approximately a 0.1v change in the output value. This thus places a limit on the resolution of the displayed value. Similarly, the corresponding output value change for the high ranges is approximately 0.18v for a 1 LSB change in the output of the A/D converters.

Similarly the current ratios using a 10 amp shunt, indicate that a 1 LSB change in the channel 2 or 4 A/D converter, results in approximately a 0.002A change in the output value. This results because the current reading is computed to two decimal places.

6.3 Performance checks

The meter was compared with a voltmeter, ammeter and digital wattmeter. As indicated in Fig. 6.1, a linear relationship between the computed rms value and input signal amplitude occured when a sinusoidal, triangular and square waveforms at 60hz were applied to the calibration input. The near constant (difference between maximum and minimum values = 5) rms values over a frequency range of 20hz to 1100hz for a sine and square wave input are indicated in Fig. 6.2. The linear relation between the wattmeter reading and the calibrated wattmeter reading are indicated for two loads in Figs. 6.3 and 6.4.

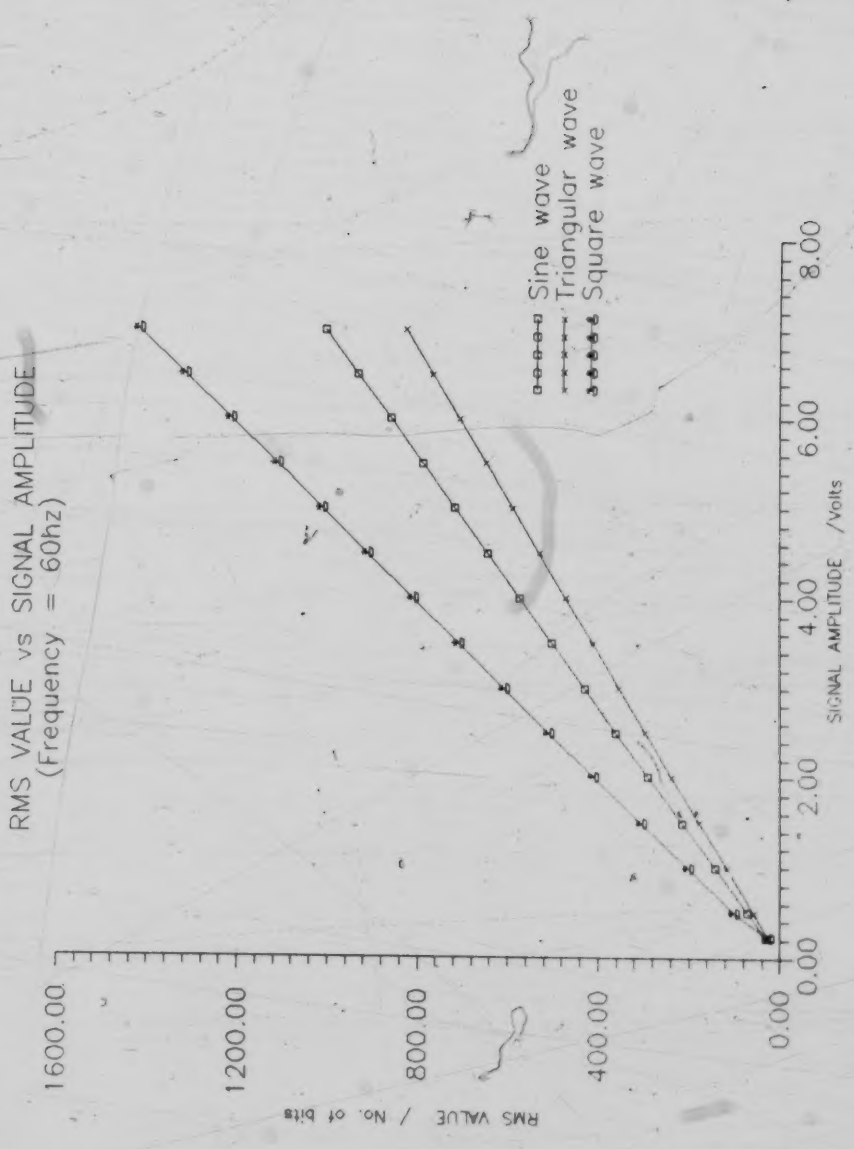


FIG. 6.1 GRAPH OF RMS VALUE vs SIGNAL FREQUENCY (60 Hz)

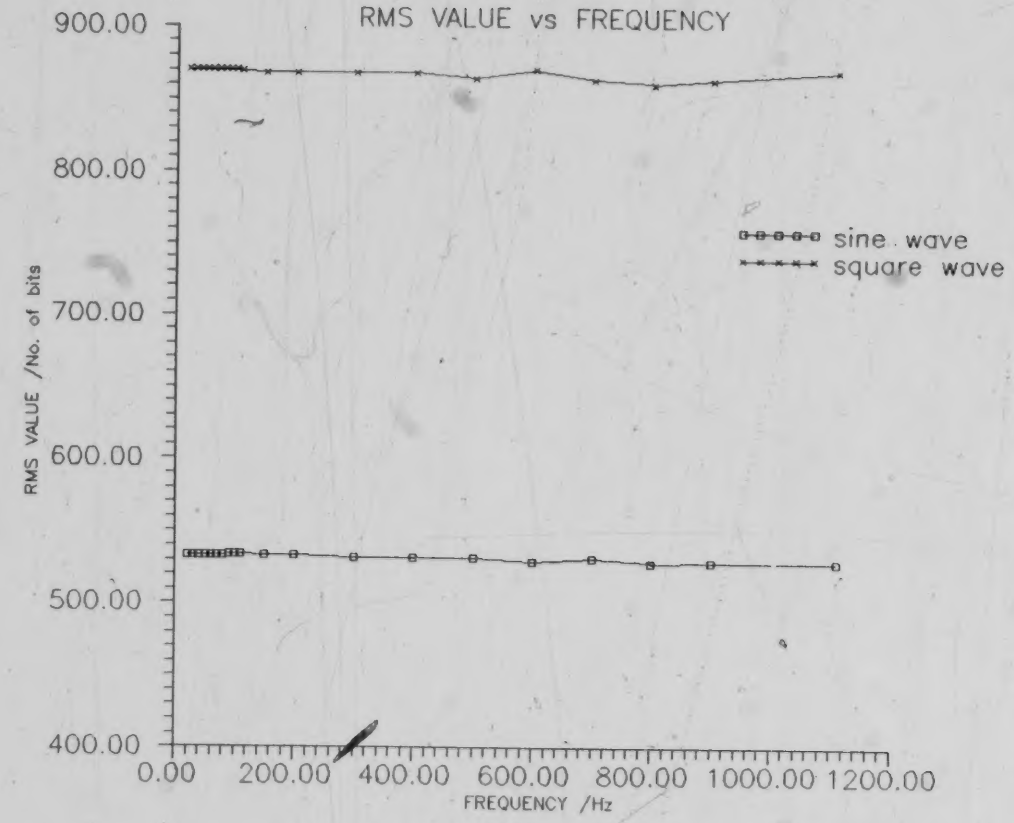


FIG. 6.2 GRAPH OF RMS VALUE vs FREQUENCY

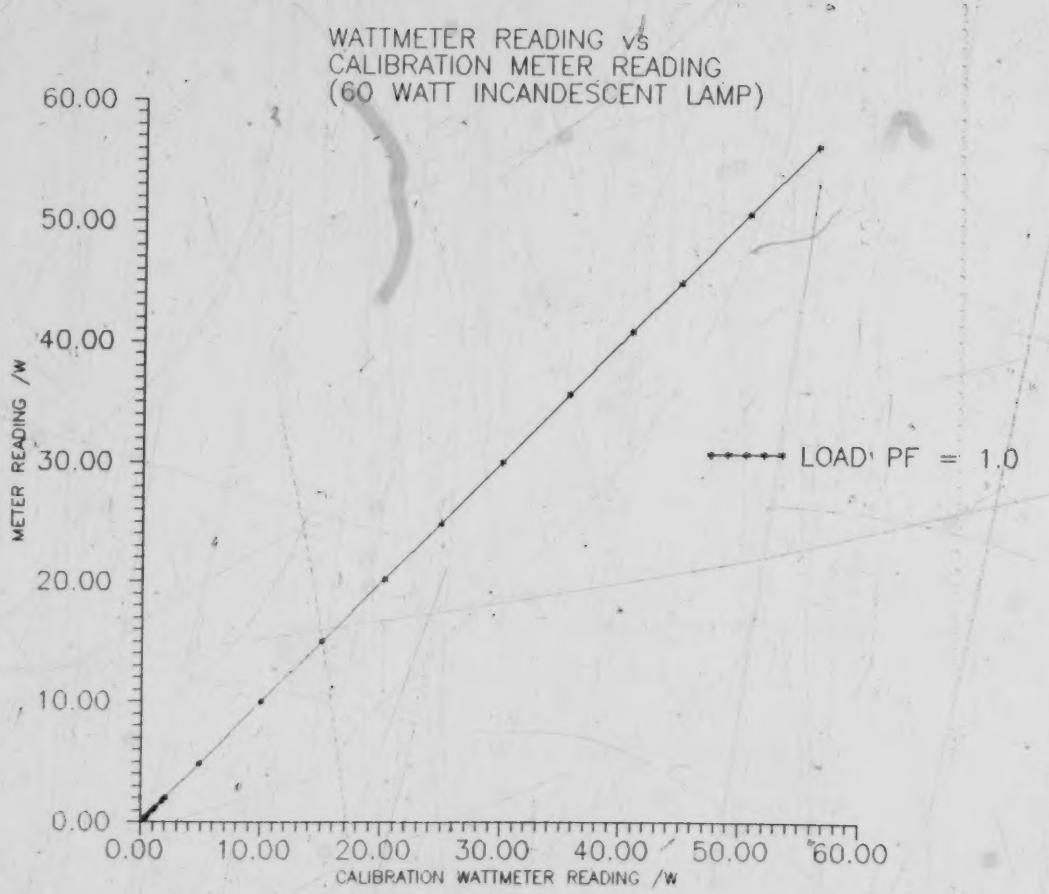


FIG. 6.3 GRAPH OF WATTMETER READING vs CALIBRATION METER READING (60 WATT INCANDESCENT LAMP)

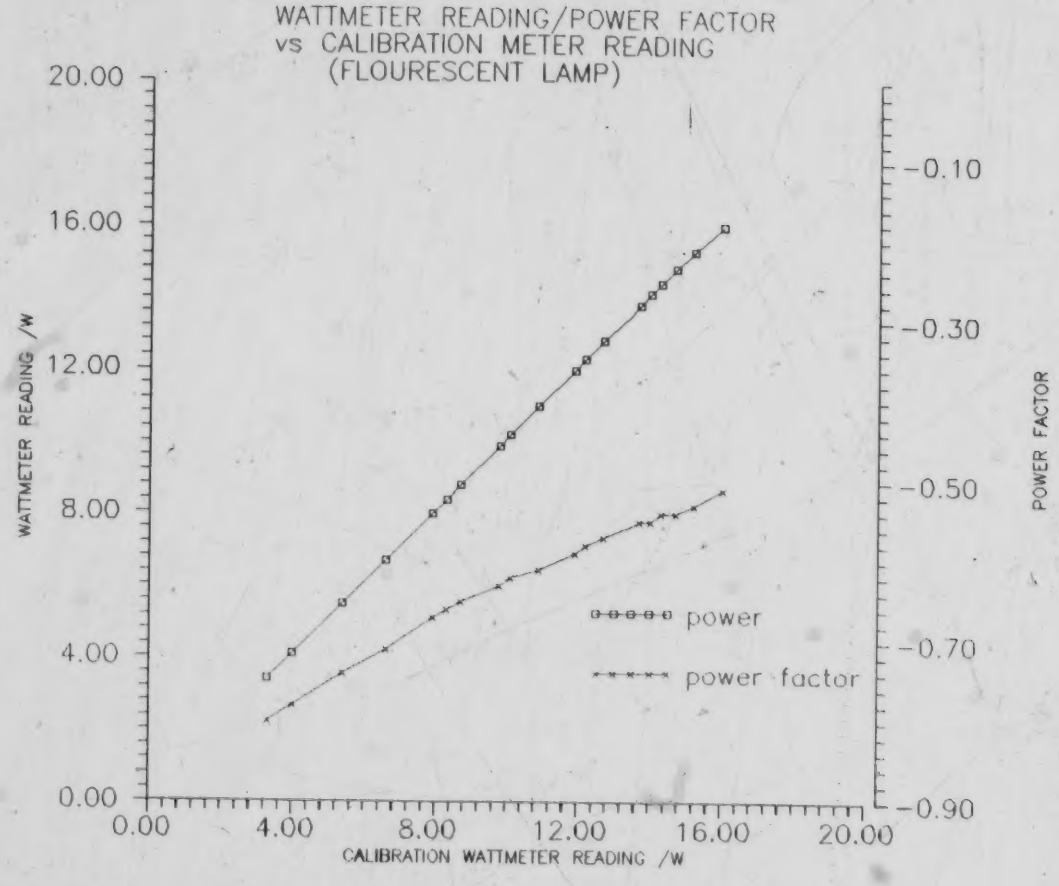


FIG. 6.4 GRAPH OF WATTMETER READING vs CALIBRATION METER READING (FLUORESCENT LAMP)

6.4 Sources of error

The significant sources of error are:

a) Noise primarily due to the switching circuits used.

b) Rounding errors in the arithmetic computations used.

c) Component tolerances especially in the analog section of the system.

The use of 1% tolerance resistors results in worst case common-mode rejection ratios of 275 and 50 for the voltage and current input conditioning circuits respectively.

6.5 Specifications

	Minimum	Maximum
Input voltage /V	-350	+350
Load current /A	-10	+10
Power /W		3500
Frequency /hz	12	1000

Time span for the acquisition, computation and display.

Mode	Maximum time
Autocalibrate(All channels sequentially)	162.1ms
DC (2 channels)	185ms
AC (1-phase)	83.5ms
AC power (3-phase)	67ms
DC power (4 channels)	187ms
Frequency (60 hz signal)	33ms

Power requirements

+5volts	1.2A

+15volts	120mA
-15volts	150mA

Voltages and power readings are read to one decimal place. Current and power factor readings can be read to two decimal places.

6.6 Suggestions for improvements

Although the system met the design specifications, finance and the available components, placed a limitation on some desireable features which could be included. A few improvements that can be made include the incorporation of devices to support a higher sampling rate. Including components with higher frequency bandwidths will enable the upper frequency limit of the instrument to be extended. In addition, an autoranging feature could also be incorporated.

7. CONCLUSION

A digital wattmeter has been designed and constructed. which compared favourably with a wattmeter with a worst case accuracy of 0.3% . Voltage measurements agreed with those obtained on a 6-digit voltmeter. An extension of the trapezoidal rule is employed. This technique, theoretically produces the best accuracy among current numerical integration techniques employed in digital sampling wattmeters. In addition to the electrical power, the voltage current and power factor are also computed. The instrument is also capable off of frequency measurement.

The microprocessor-based instrument can measure the parameters of two loads simultaneously. The hardware uses off-the-shelf components. A GPIB port is provided for integrating the instrument into an automated test setup.

The system software consists of a number of subprograms which are used in acquiring, storing, computing and displaying various parameters and the system status.

The outstanding features of this thesis are:

(a) the first practical implementation of an instrument employing the modified trapezoidal method.

(b) the instrument is easy to use

(c) software is optimized for speed and size

(d) reasonably priced ($500)

REFERENCES

1. CLARKE F. J. J., STOCKTON J. R. "Principles and theory of wattmeters operating on the basis of regularly spaced sample pairs", **Journal of Physics. E, Scientific Instruments**, Vol. 15. 1982, pp. 645-652.
2. CORNEY A.C., PULLMAN R.T. "Digital Sampling Laboratory Wattmeter", **IEEE Transactions on Instrumentation and Measurement, Vol. IM-36, No. 1**, March 1987, pp. 54-59.
3. DEKKER R., BEENKER F., THIJSSEN L. "Fault Modeling and Test Algorithm Development for Static Random Access Memories" **IEEE 1988 International Test Conference,** pages 343-352.
4. FRANCO S., Design with Operational Amplifiers and Analog Integrated Circuits, **McGraw-Hill Book Company**,1988.
5. GREGORY B.A. An Introduction to Electrical Instrumentation, **Macmillan**, 1973.
6. HILL J. J., ANDERSON W. E., "Design of a microprocessor-based digital wattmeter", **IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. IECI-28, No. 3**, August 1981, pp. 180-184.
7. INTEL MICROPROCESSOR AND PERIPHERAL HANDBOOK, VOL. II, PERIPHERAL , 1988.
8. MATOUKA M.F., "A wide-range digital power/energy meter for systems with nonsinusoidal waveforms." **IEEE**

Transactions on Industrial Electronics, Vol. IE-36, No. 1, February 1982, pp. 18-31.

9. MOTOROLA MICROPROCESSORS DATA MANUAL ,1981.
10. RATHORE T. S. "Theorems on power, mean and RMS values of uniformly sampled periodic signals" **IEE PROCEEDINGS, Vol. 131, Pt. A. No. 8,** November 1984, pp. 598-600.
11. SAY M.G., Electrotechnology Basic Theory and Circuit Calculations for Electrical Engineers, **Newnes-Butterworth,1974.**
12. STENBAKKEN G. N., "A wideband sampling wattmeter," **IEEE Trans. Power App. Syst.,** Vol PAS-103, no. 10 Oct. 1984, pp. 2919-2926.
13. TURGEL R. S. "Digital Wattmeter Using a Sampling Method," **IEEE Trans. on Instrumentation and Measurement**, Vol. IM-23, No. 4, Dec. 1974, pp. 337-341.
14. ZU-LIANG L., "An Error Estimate for Quasi-Integer-Period Sampling and an Approach for Improving its Accuracy", **IEEE Trans. on Instrumentation and Measurement,** Vol. 37. No. 2. June 1988, pp. 219-222.

APPENDIX A WATTMETER CIRCUIT DIAGRAMS

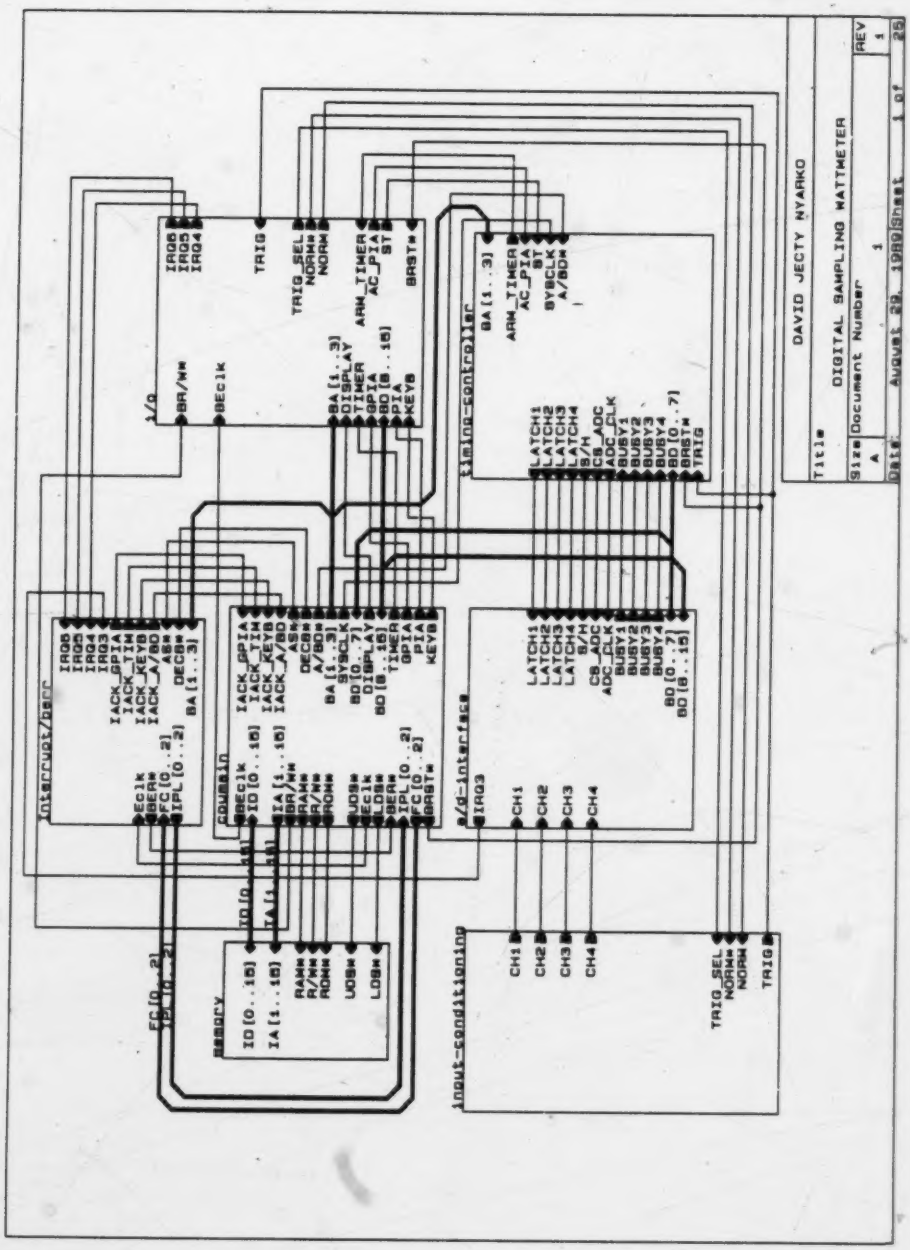


FIG. A.1 BLOCK DIAGRAM OF DIGITAL SAMPLING WATTMETER

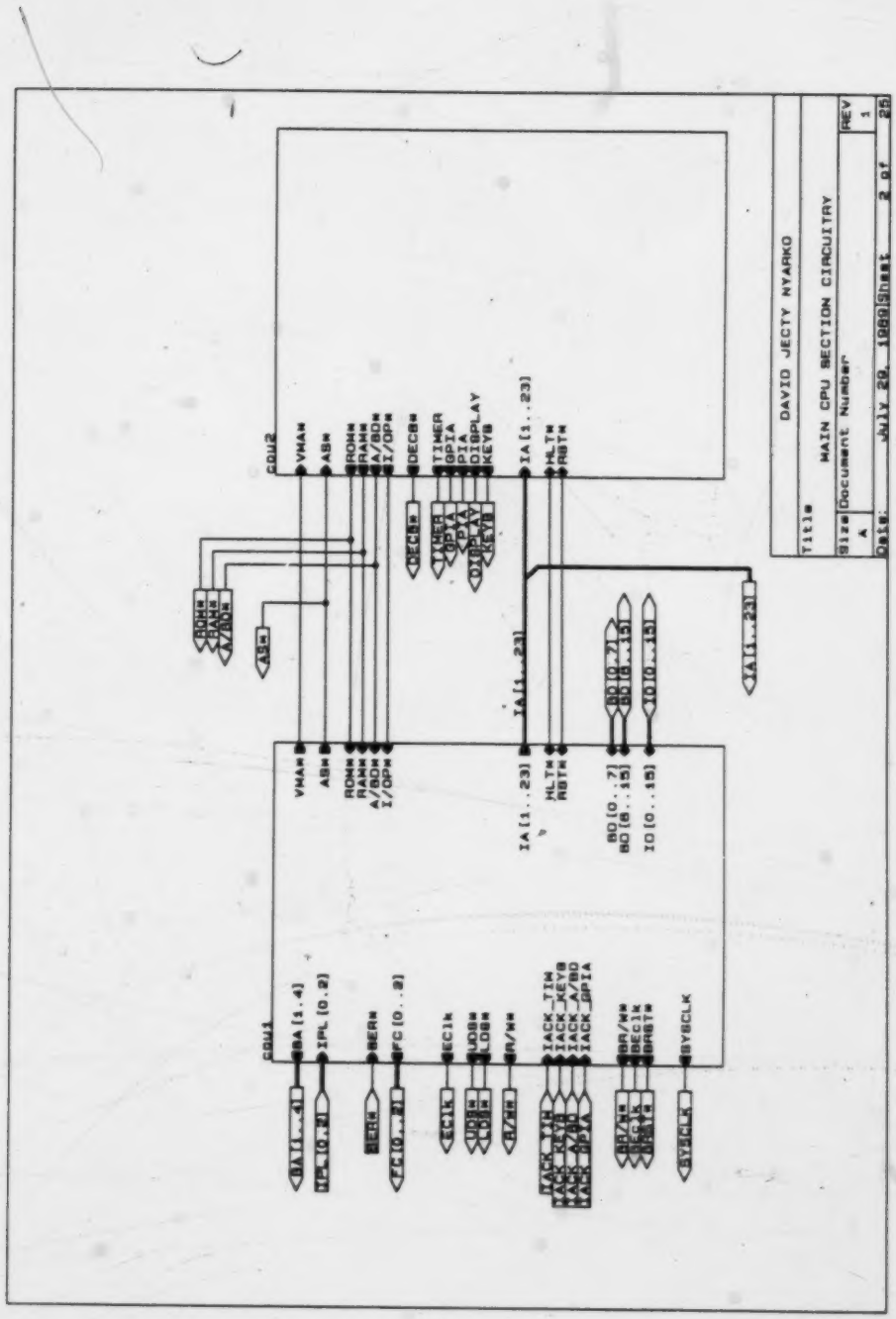


FIG. A.2 BLOCK DIAGRAM OF MAIN CPU CIRCUITRY

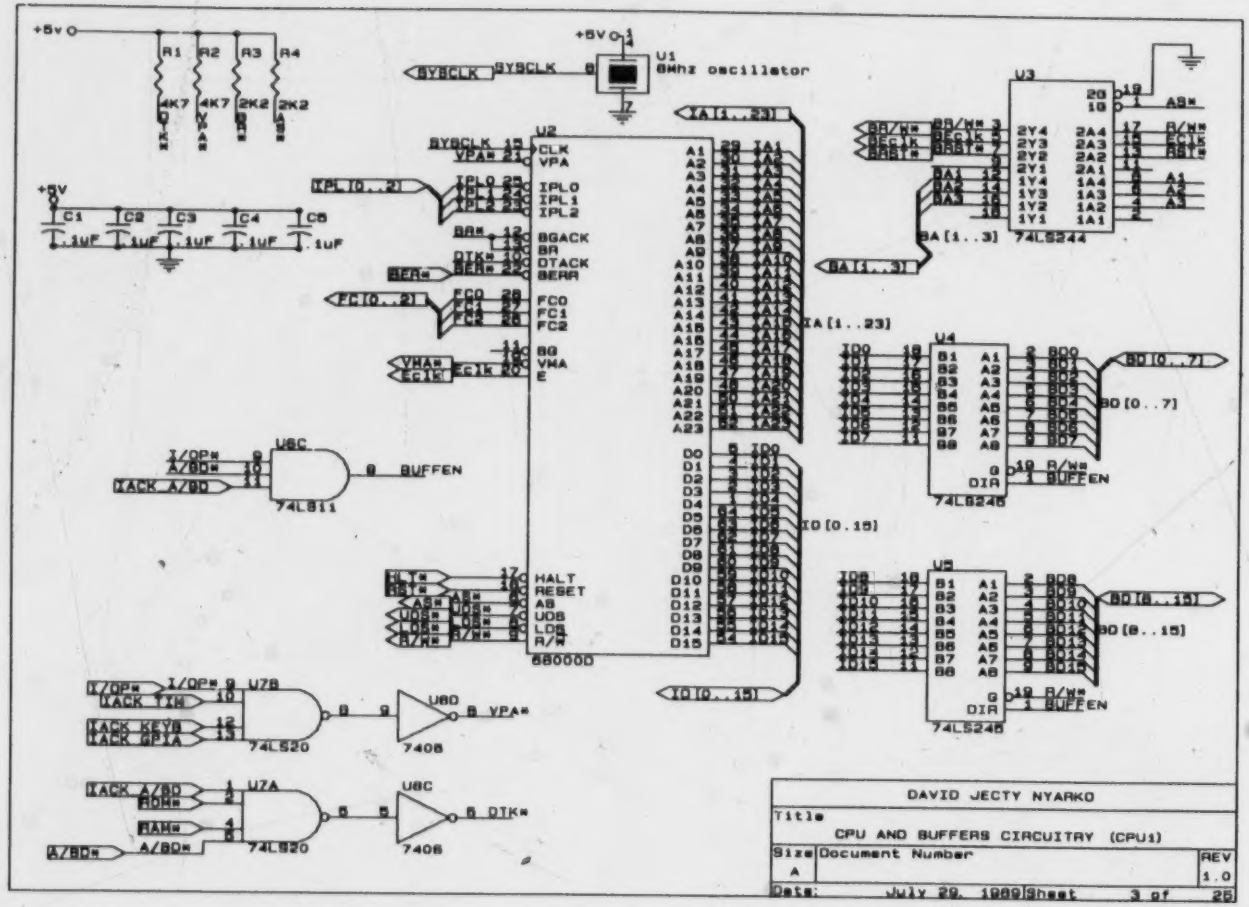


FIG. A.3 DETAILED SCHEMATIC OF CPU AND BUFFERS CIRCUITRY

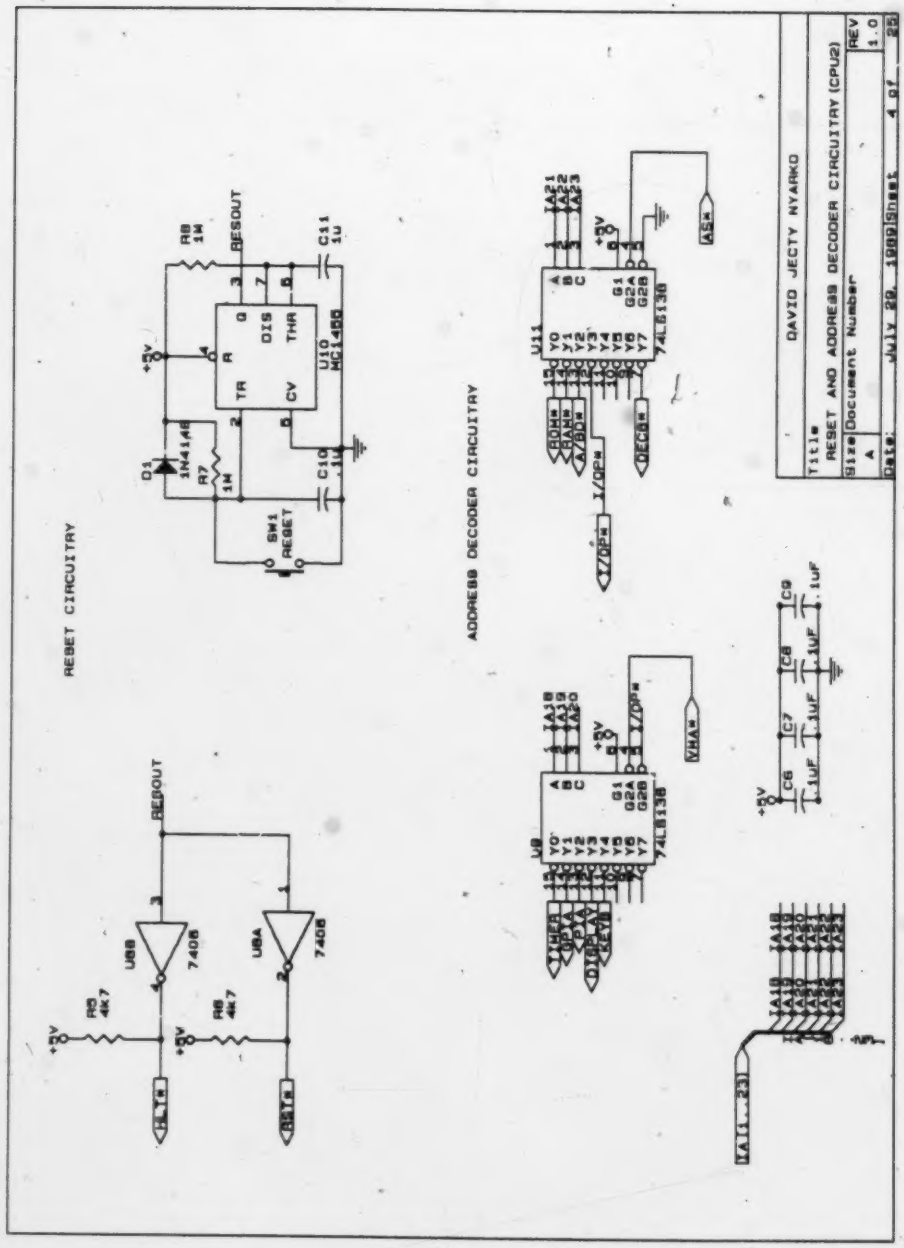


FIG. A.4 DETAILED SCHEMATIC OF RESET AND ADDRESS DECODER CIRCUITRY

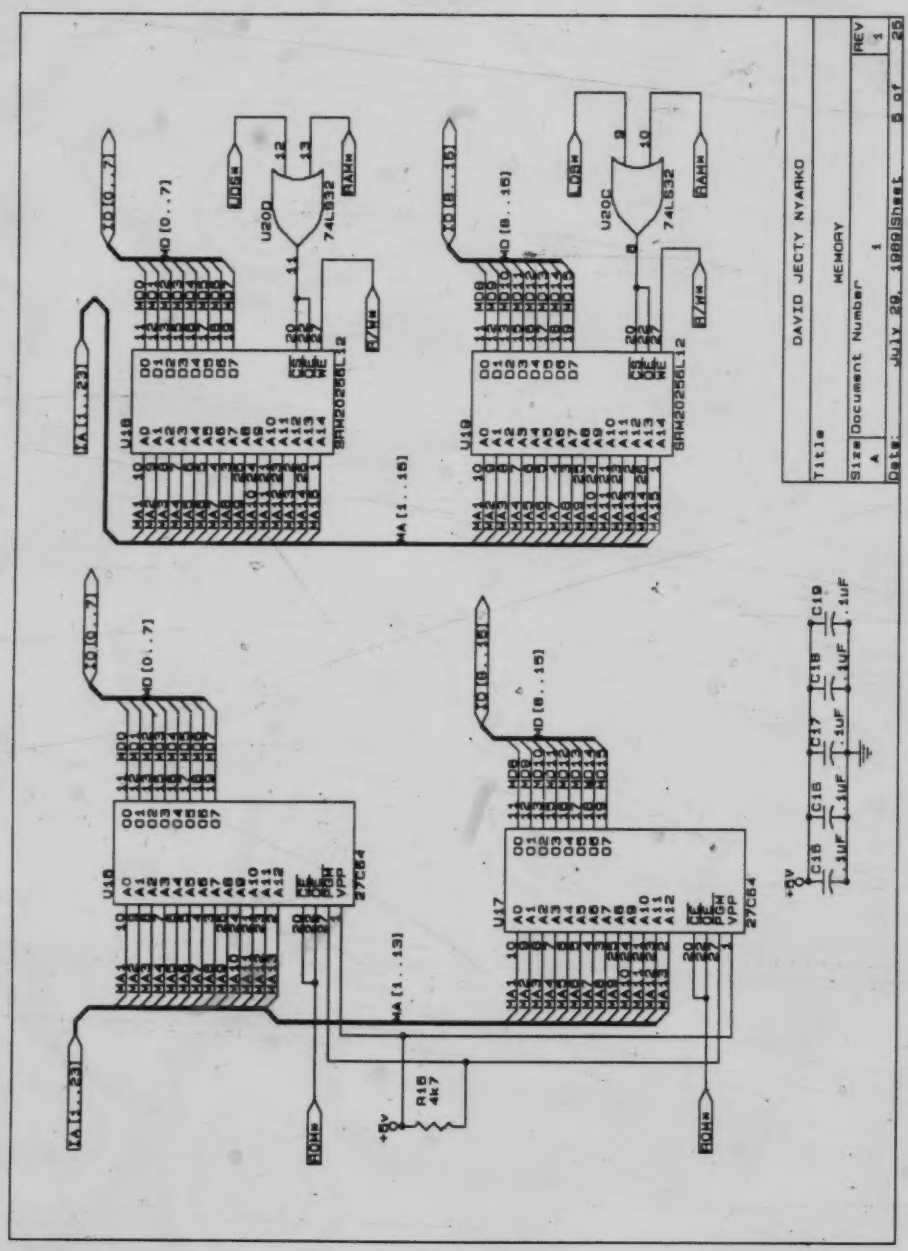


FIG. A.5 DETAILED SCHEMATIC OF MEMORY CIRCUITRY

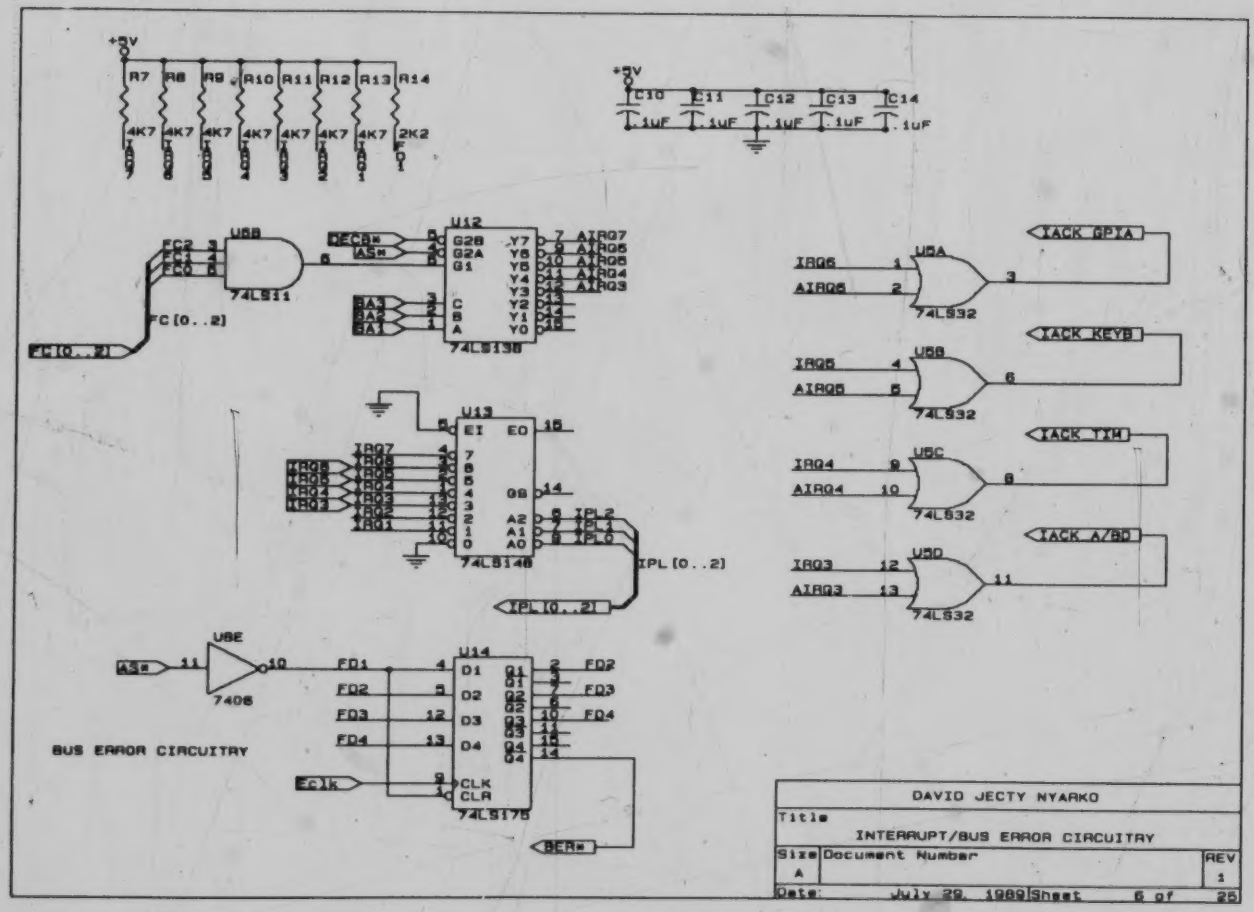


FIG. A.6 DETAILED SCHEMATIC OF INTERRUPT/BUS ERROR CIRCUITRY

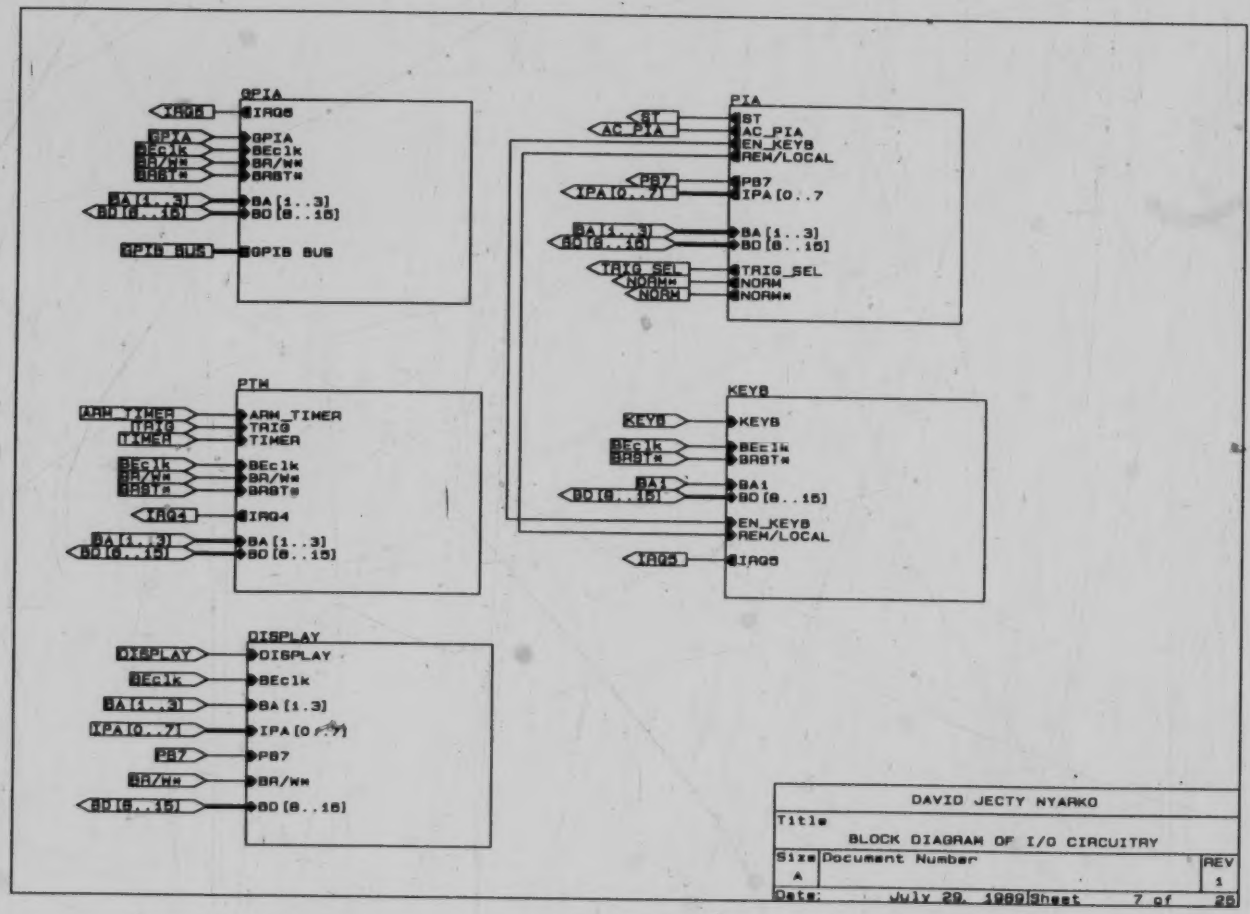


FIG. A.7 BLOCK DIAGRAM OF I/O CIRCUITRY

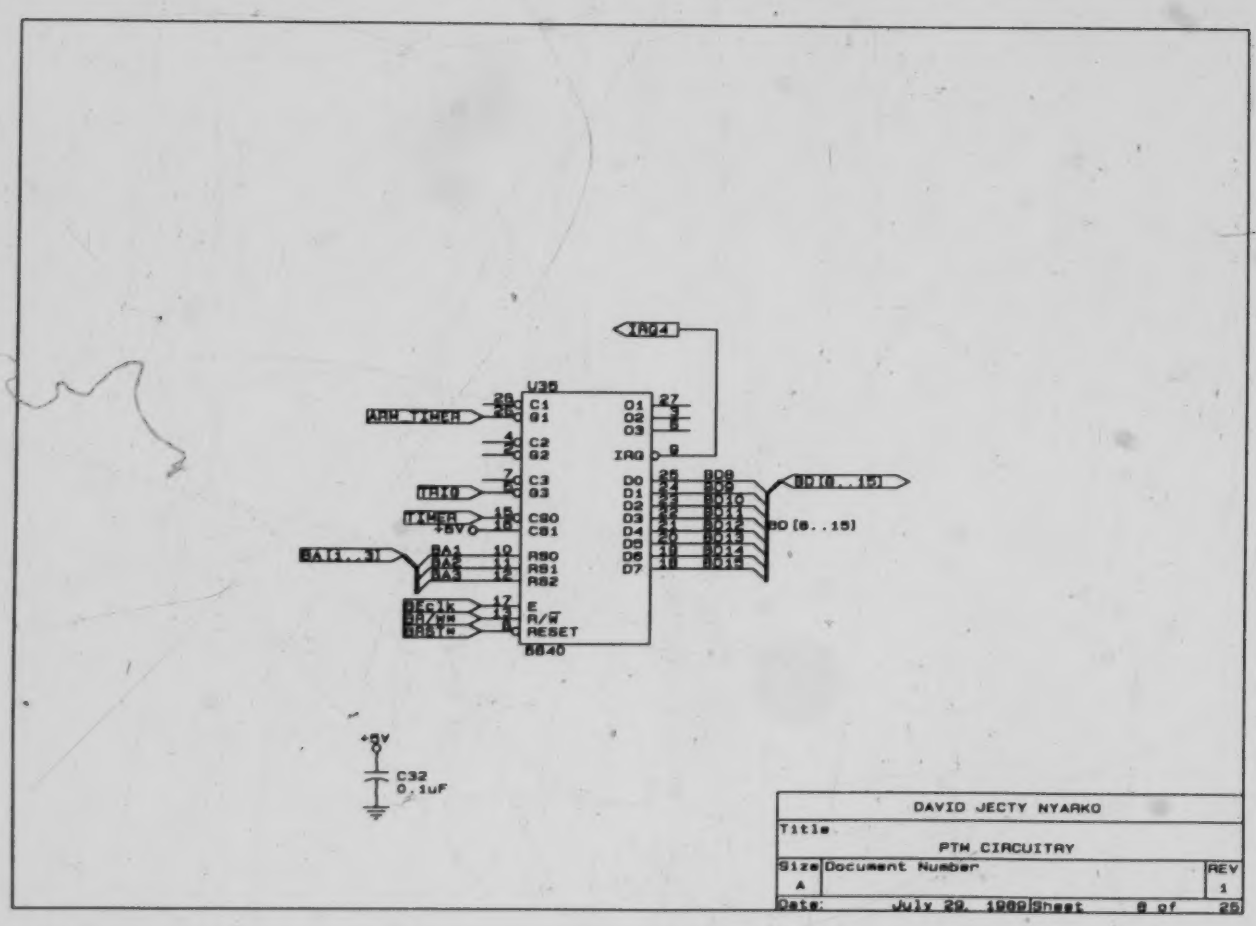


FIG. A.8 DETAILED SCHEMATIC OF PTM CIRCUITRY

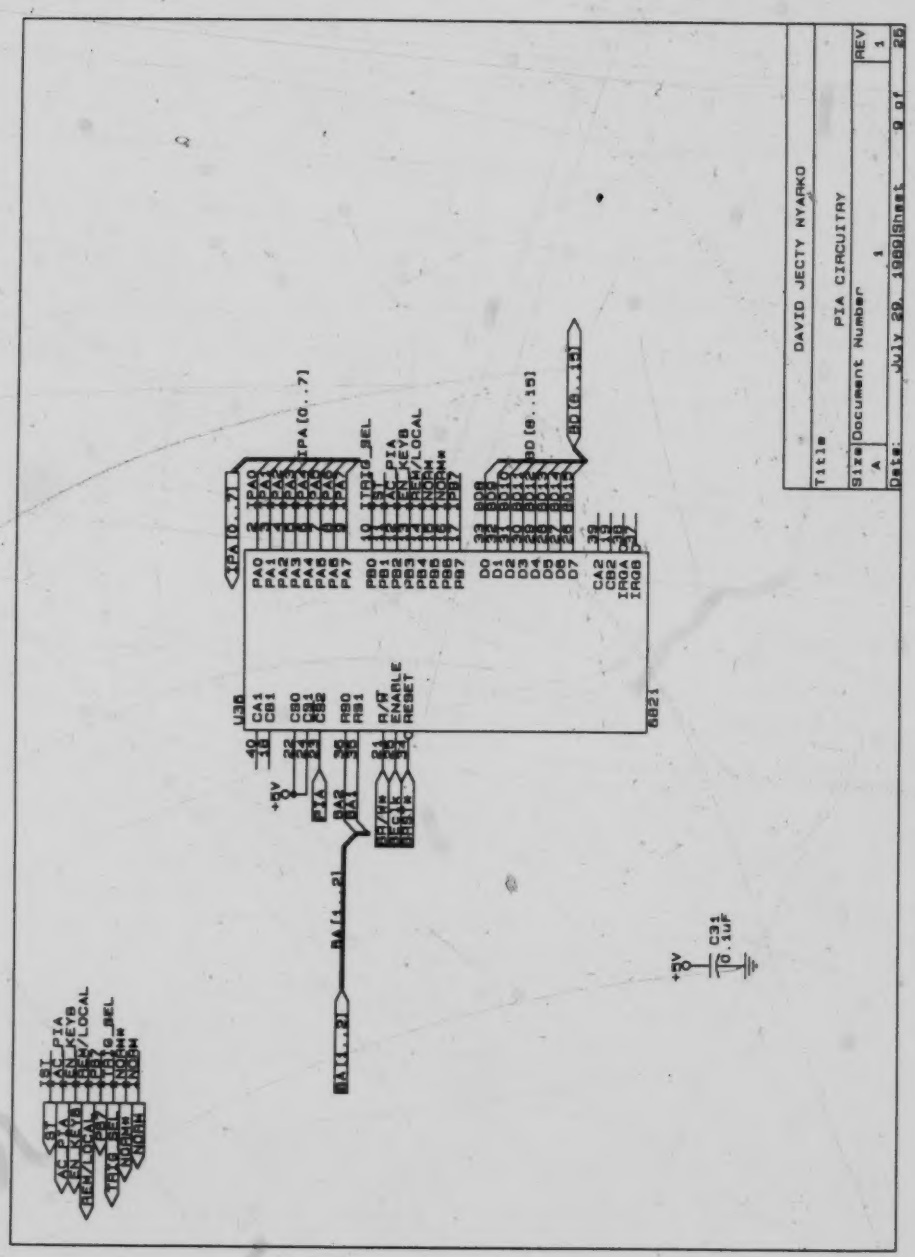


FIG. A.9 DETAILED SCHEMATIC OF PIA CIRCUITRY

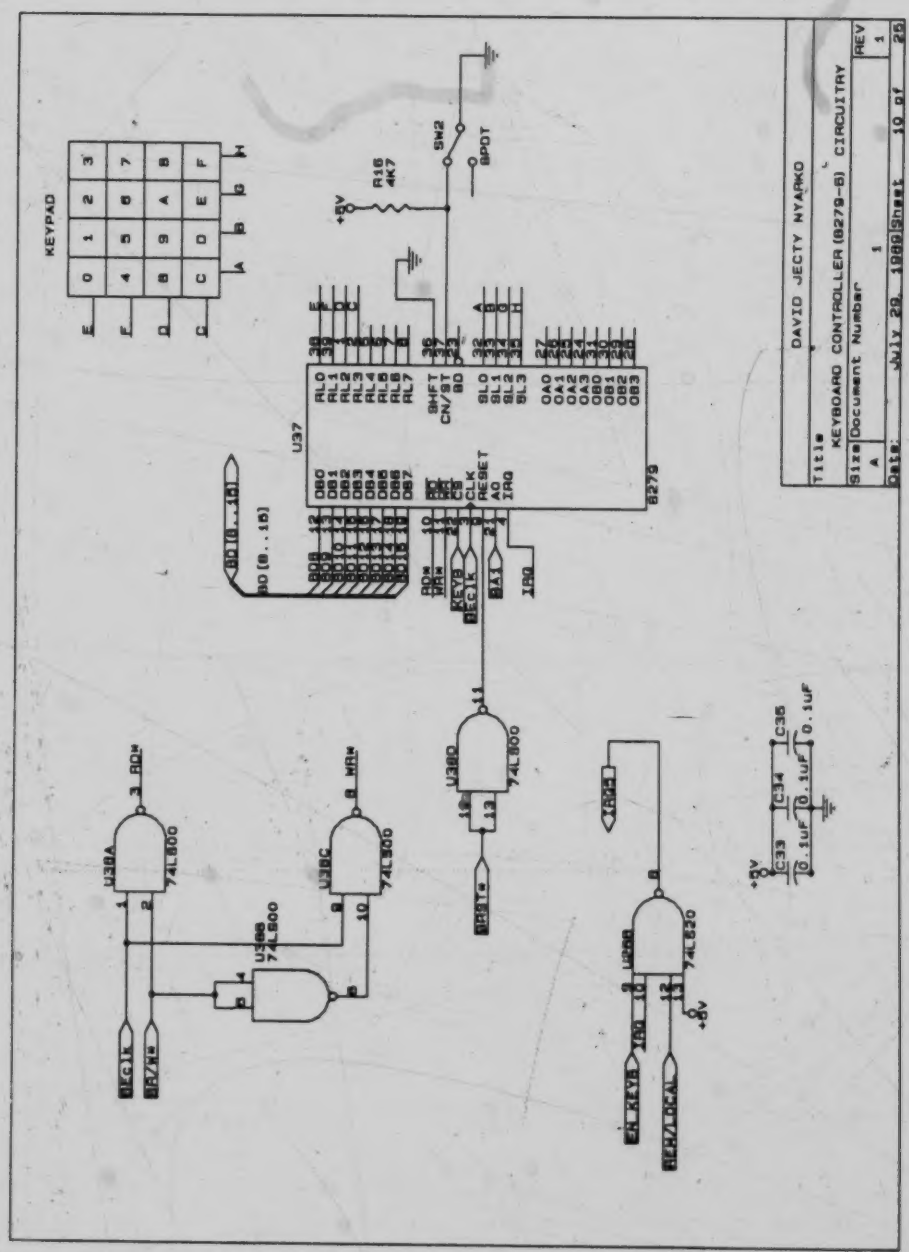


FIG. A.10 DETAILED SCHEMATIC OF KEYBOARD CONTROLLER CIRCUITRY

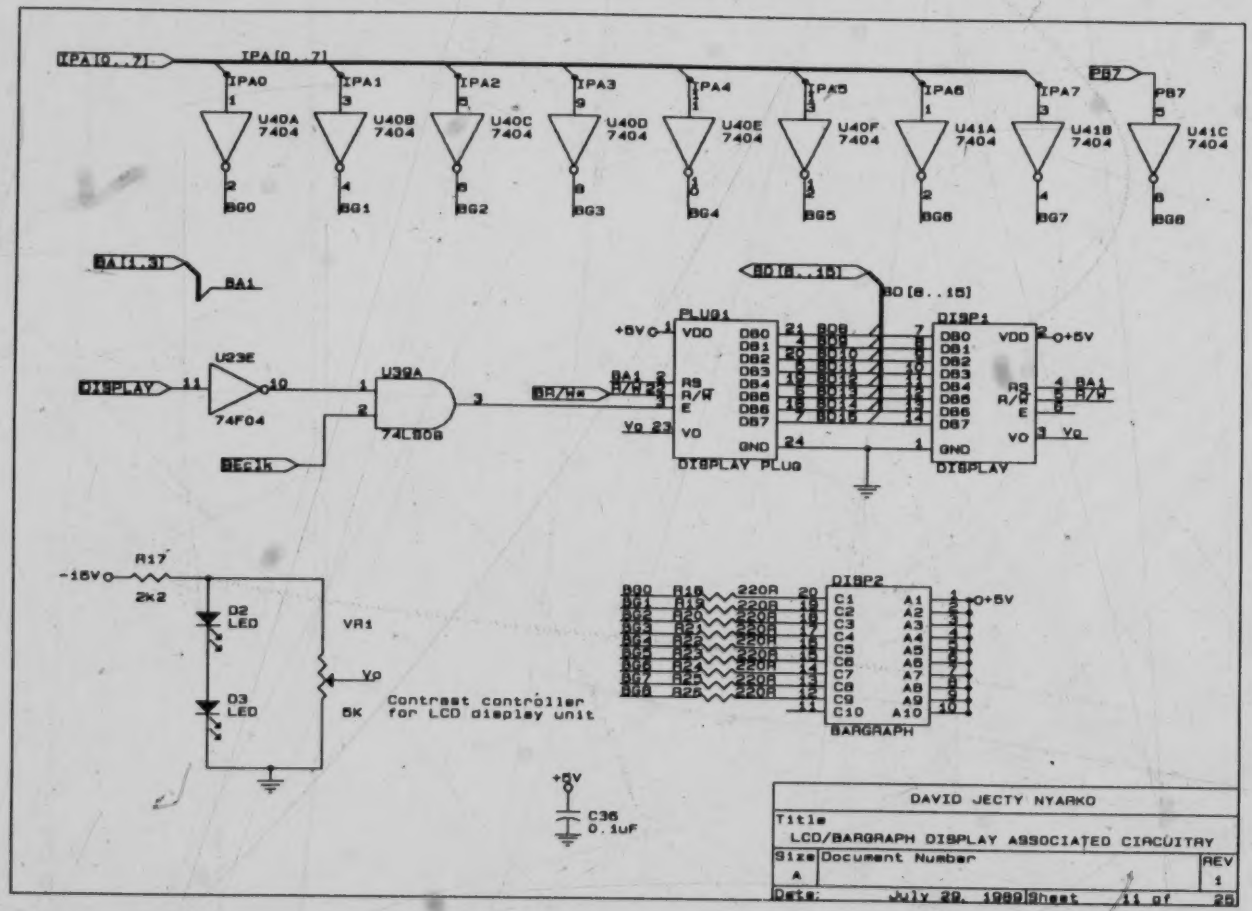


FIG. A.11 DETAILED SCHEMATIC OF LCD/BARGRAPH DISPLAY AND ASSOCIATED CIRCUITRY

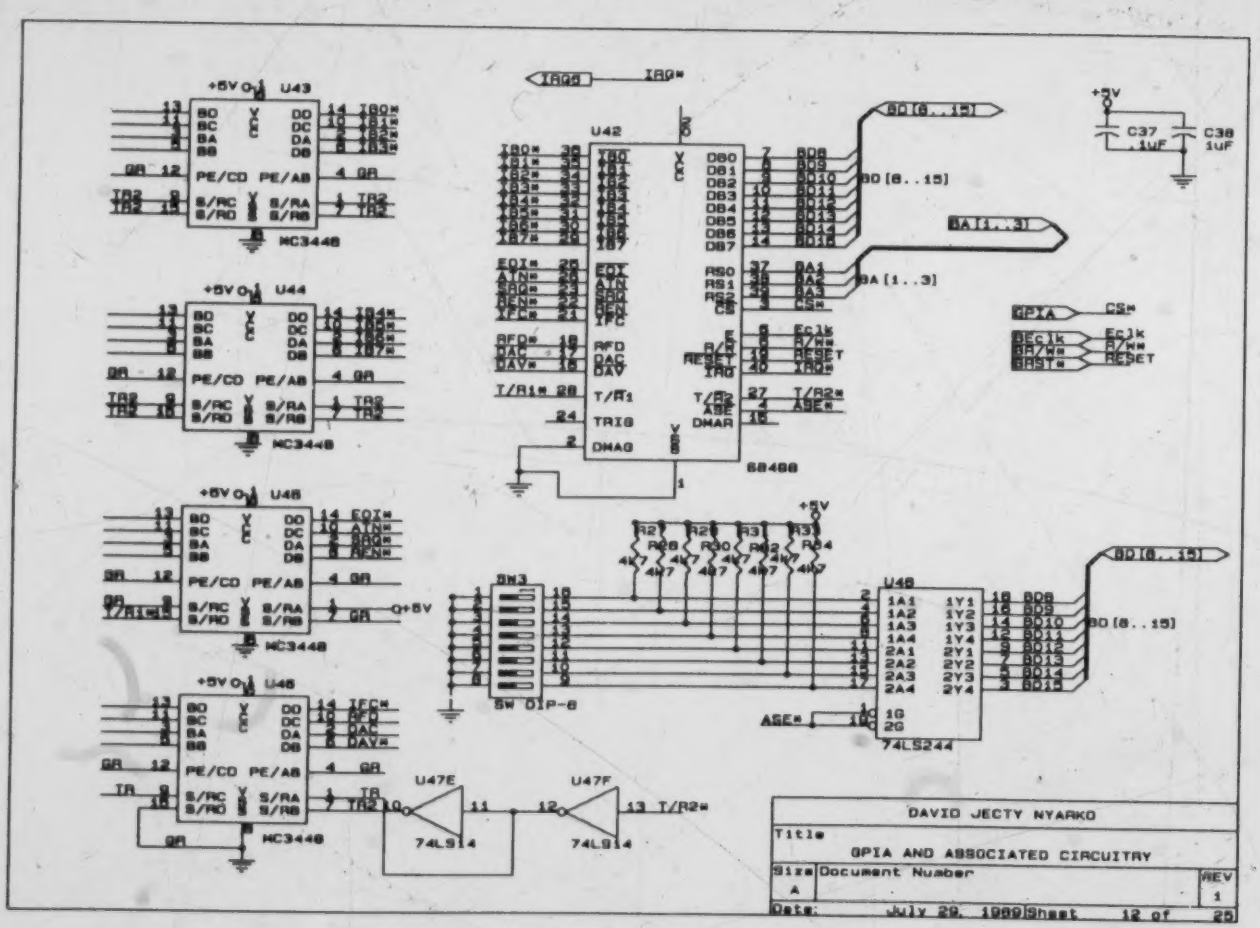


FIG. A.12 DETAILED SCHEMATIC OF GPIA AND ASSOCIATED CIRCUITRY

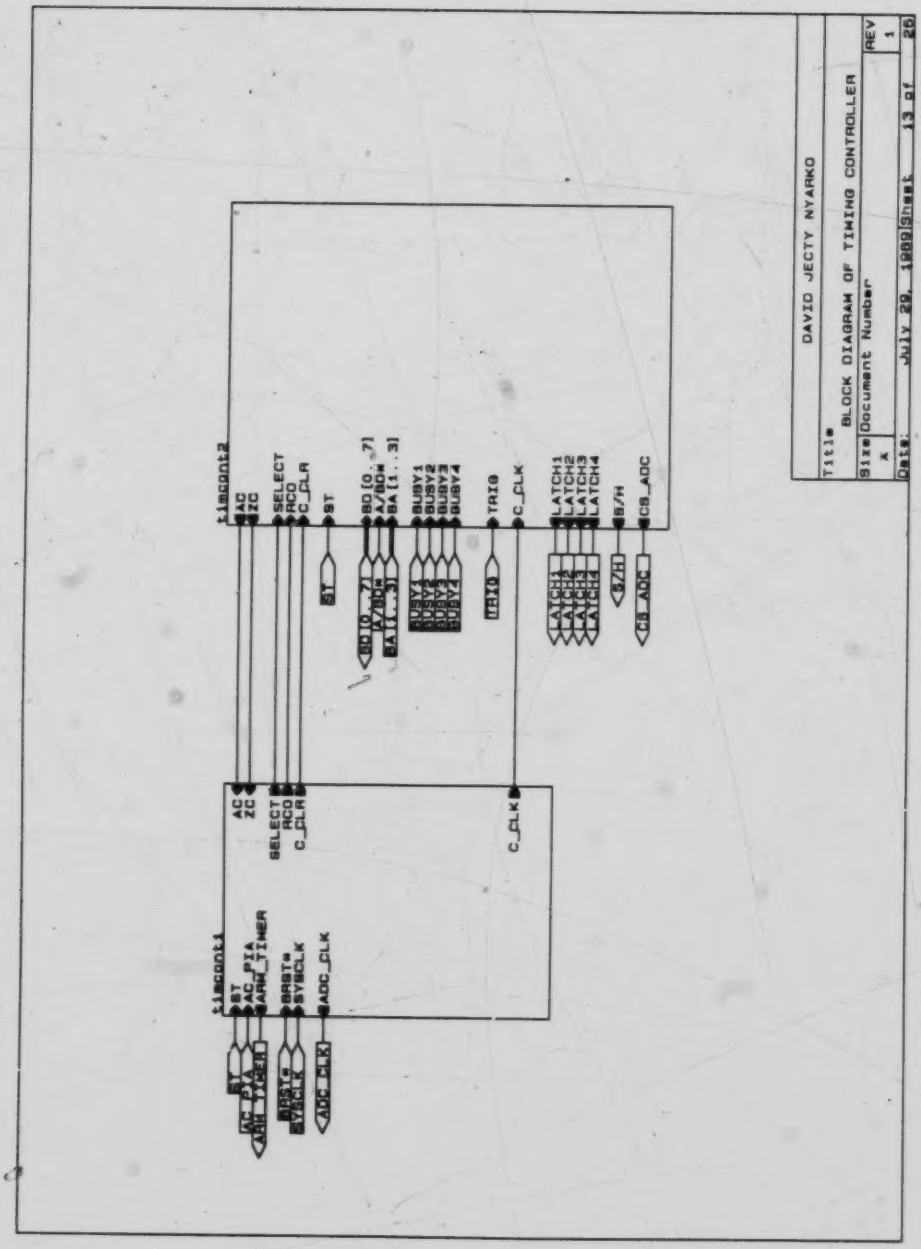


FIG. A.13 BLOCK DIAGRAM OF TIMING CONTROLLER

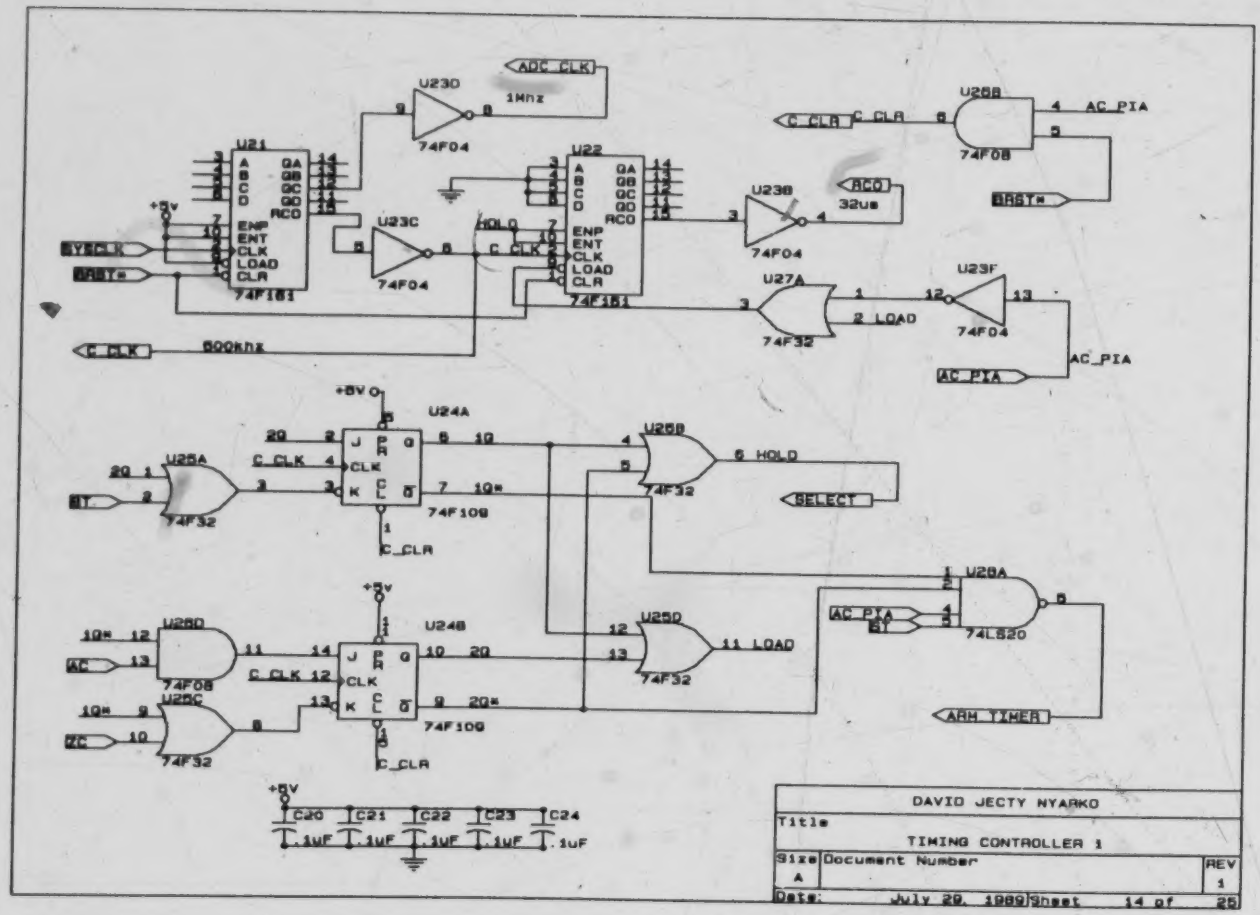


FIG. A.14 DETAILED SCHEMATIC OF TIMING CONTROLLER 1 CIRCUITRY

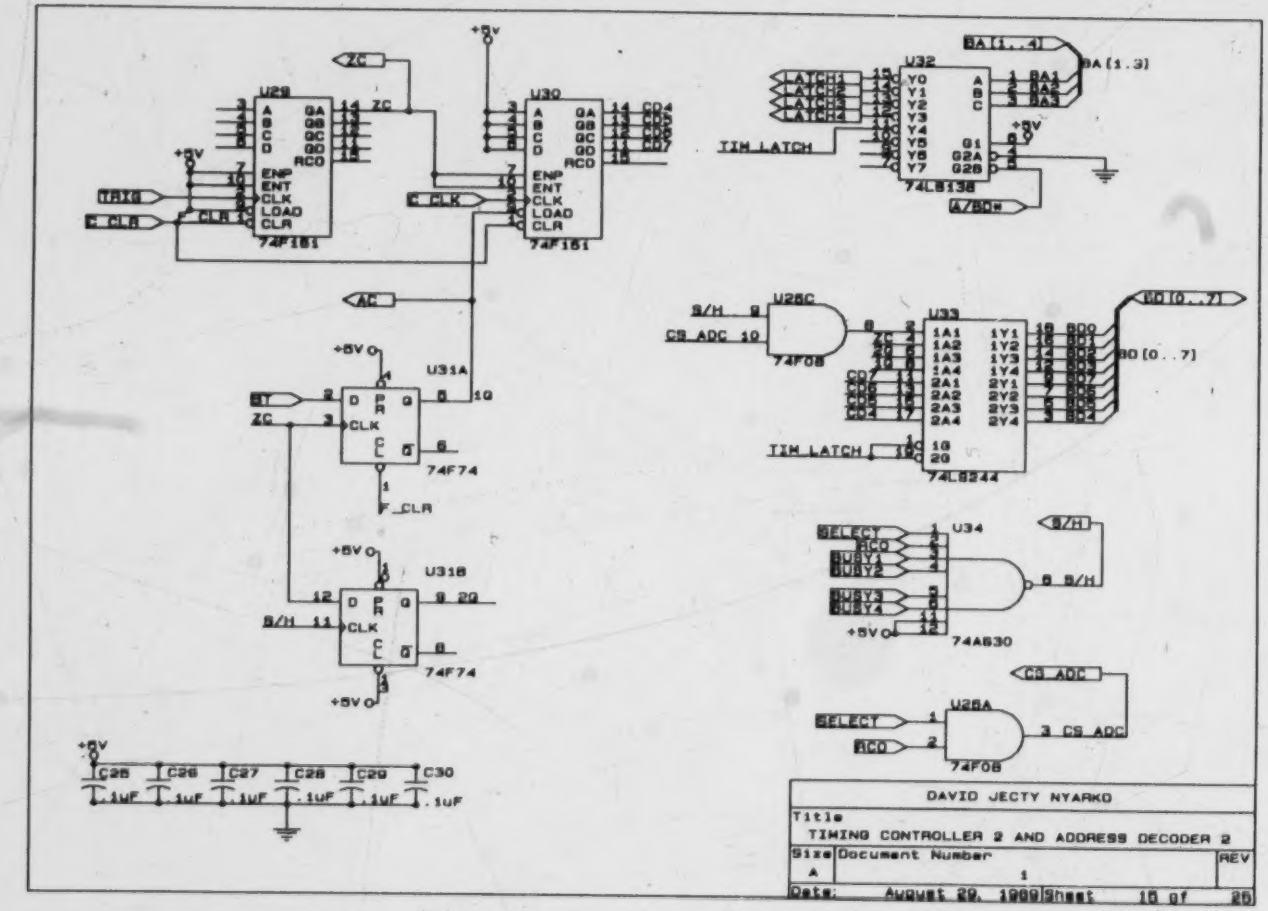


FIG. A.15 DETAILED SCHEMATIC OF TIMING CONTROLLER 2 AND ADDRESS DECODER 2 CIRCUITRY

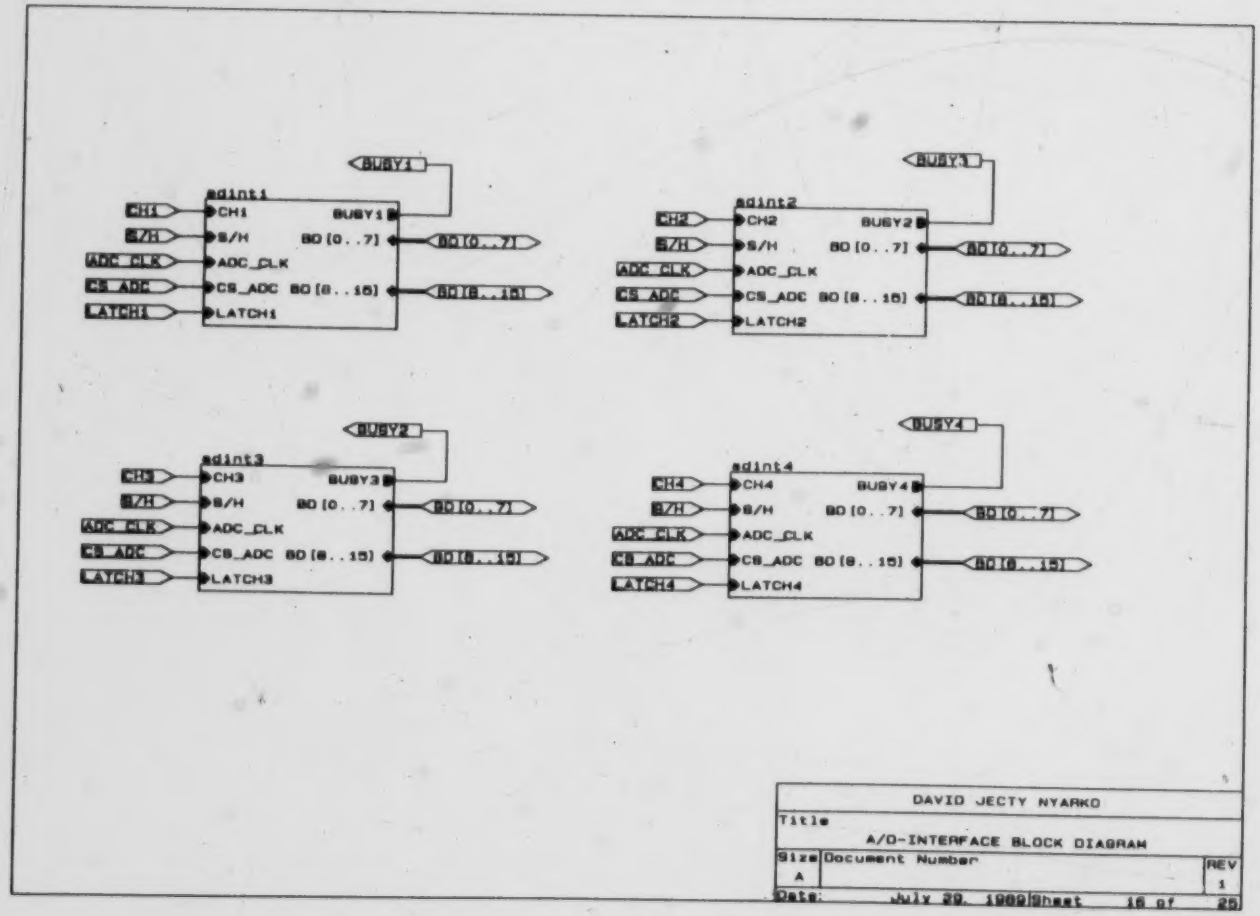
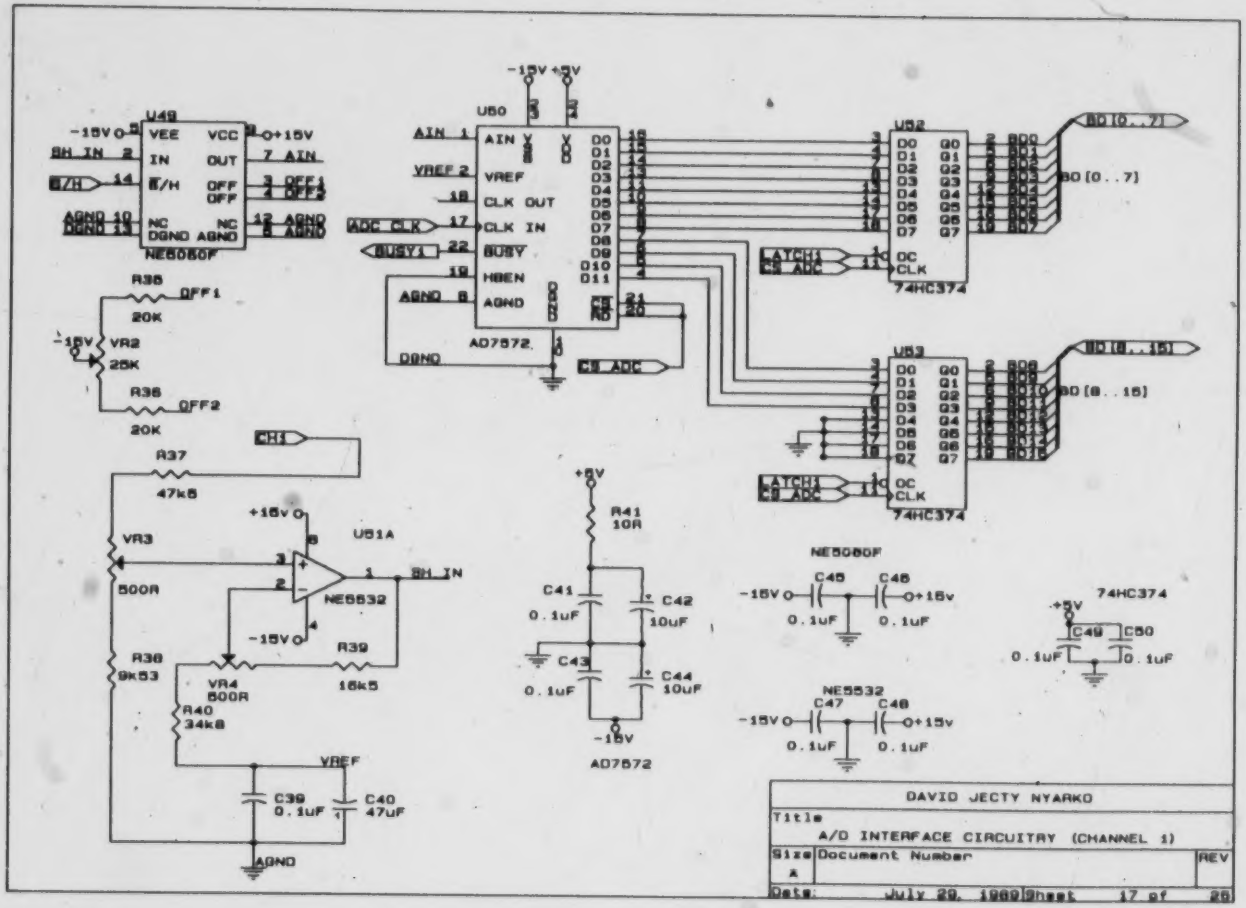


FIG. A.16 BLOCK DIAGRAM OF THE A/D INTERFACE CIRCUITRY



DAVID JECTY NYARKO

Title: A/D INTERFACE CIRCUITRY (CHANNEL 1)

Size: A | Document Number | REV

Date: July 29, 1989 | Sheet 17 of 26

FIG. A.17 DETAILED SCHEMATIC OF THE A/D INTERFACE CIRCUITRY (channel 1)

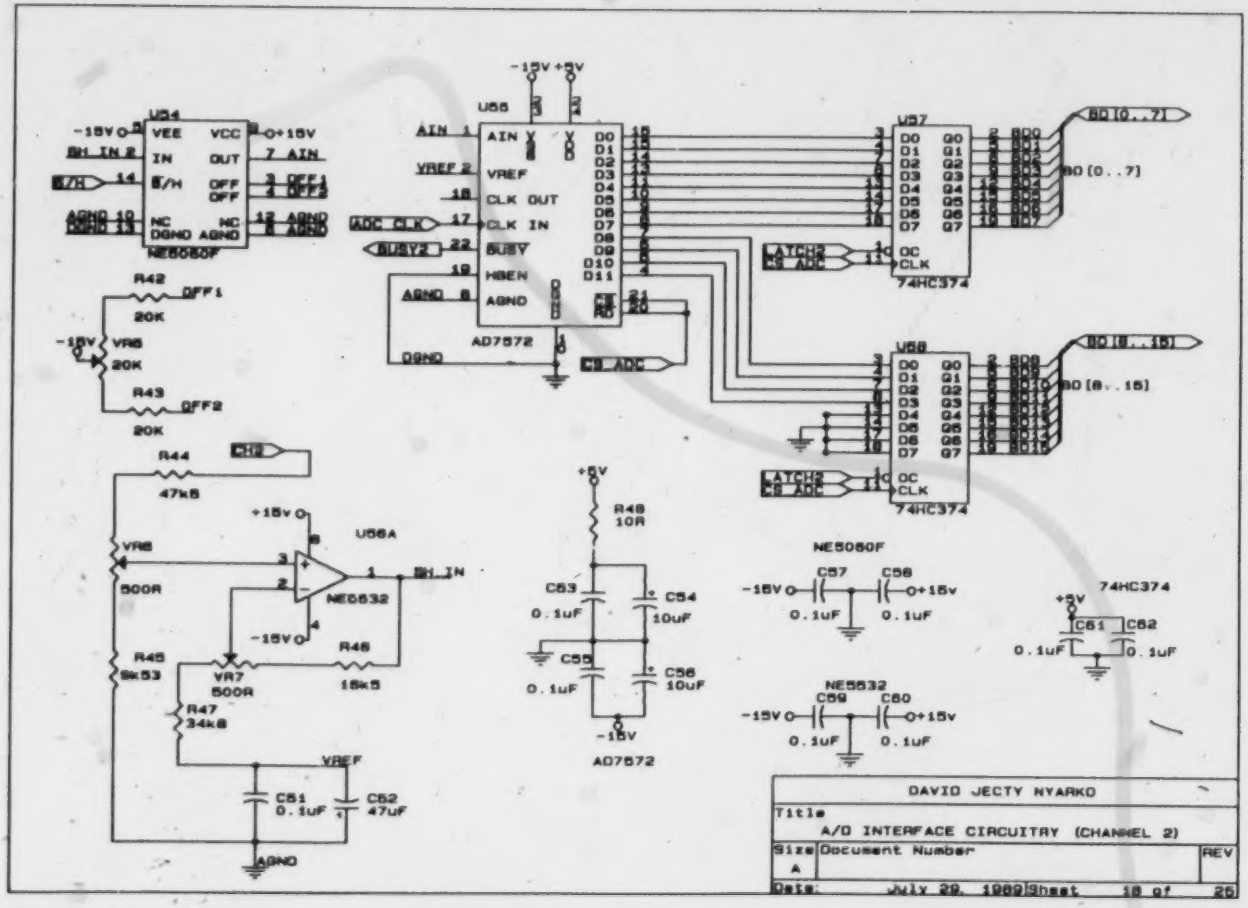


FIG. A.18 - DETAILED SCHEMAATIC OF THE A/D INTERFACE CIRCUITRY (channel 2)

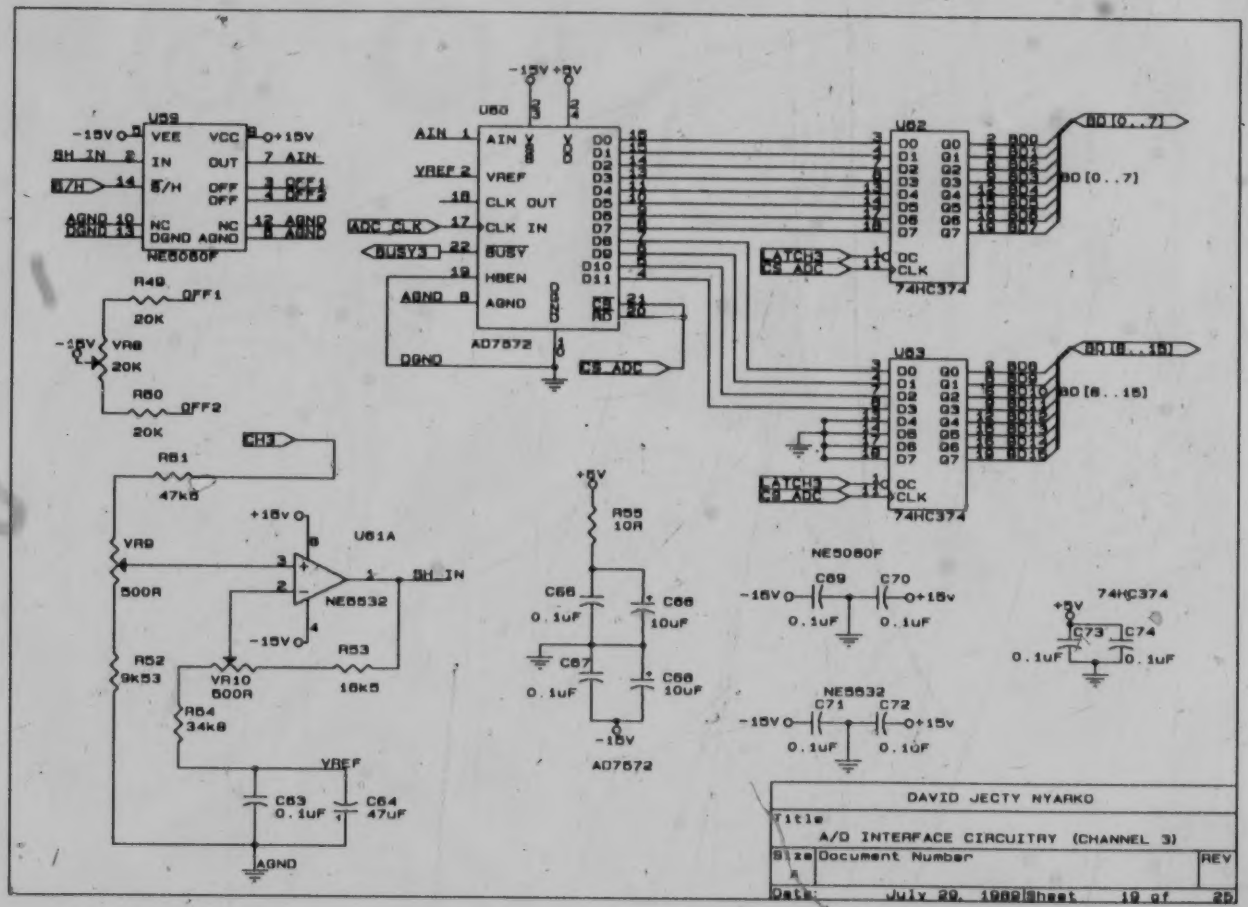


FIG. A.19 DETAILED SCHEMATIC OF THE A/D INTERFACE CIRCUITRY (channel 3)

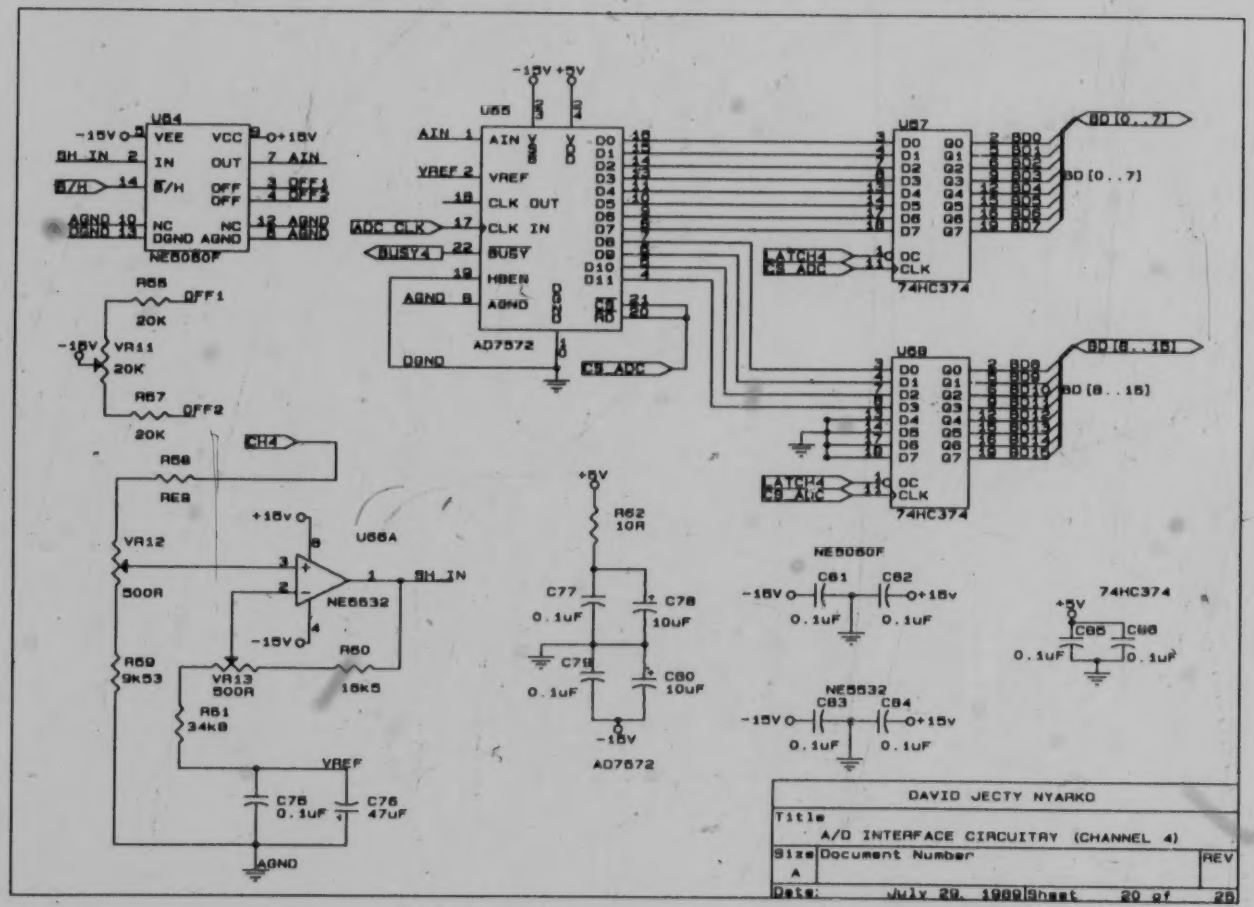


FIG. A.20 DETAILED SCHEMATIC OF THE A/D INTERFACE CIRCUITRY (channel 4)

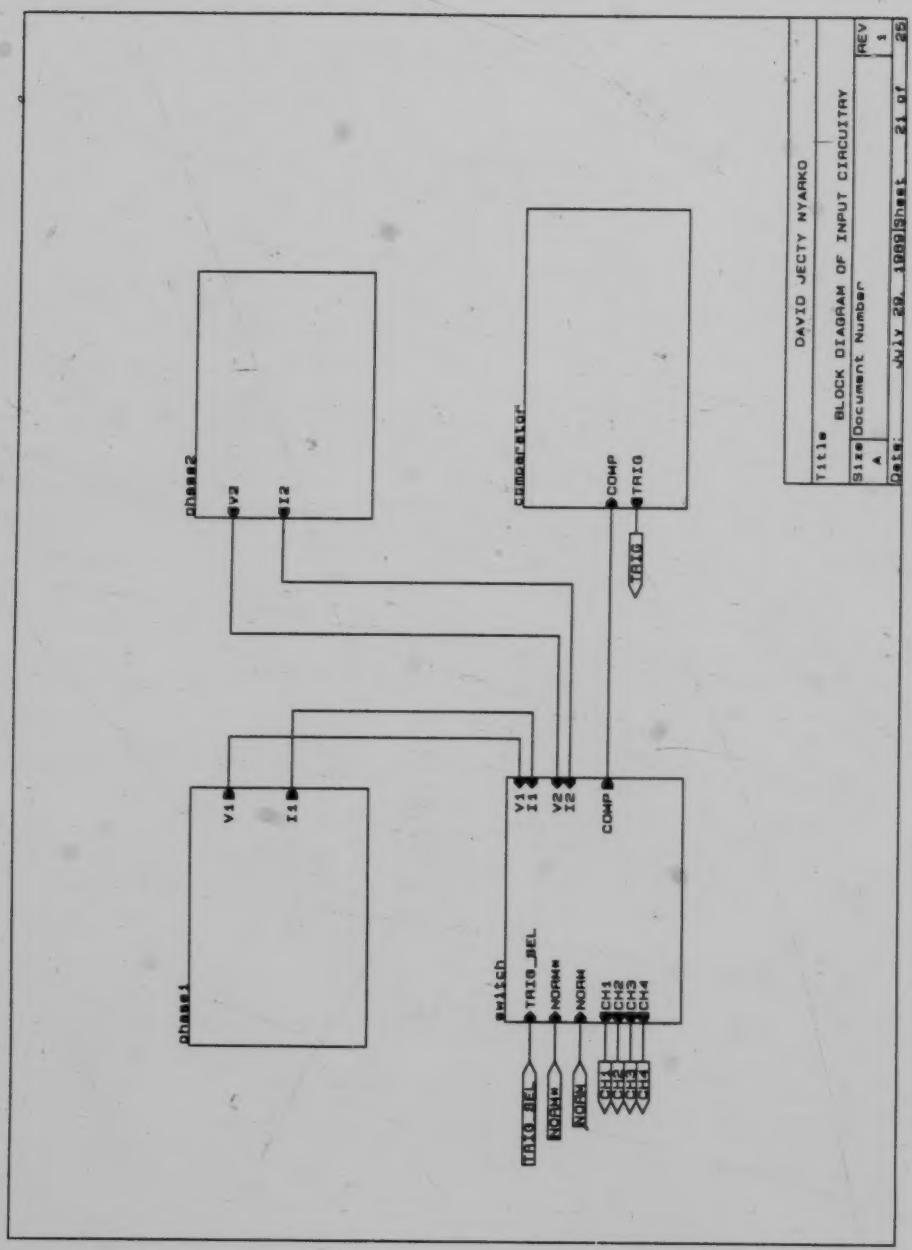


FIG. A.21 BLOCK DIAGRAM OF THE INPUT-CONDITIONING CIRCUITRY

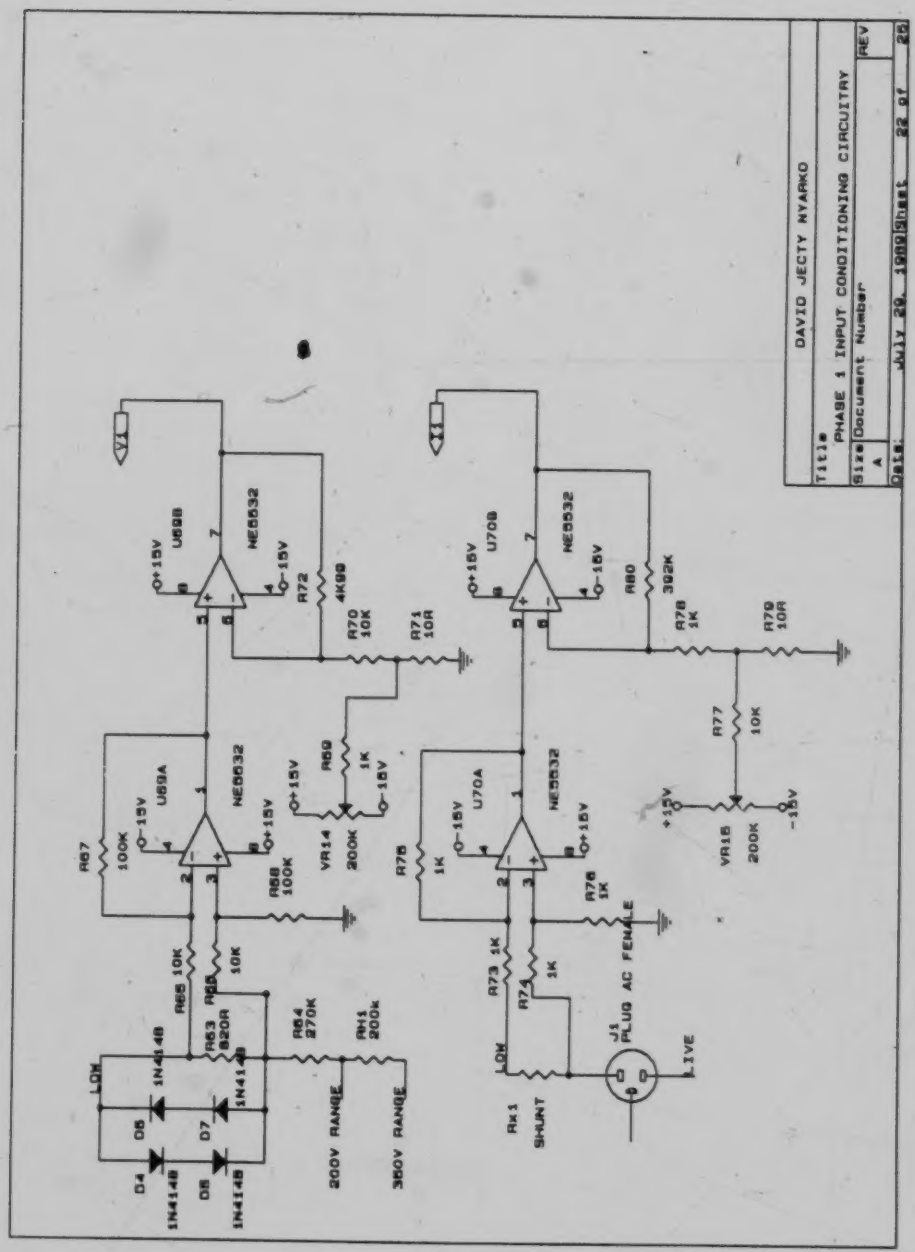


FIG. A.22 DETAILED SCHEMATIC OF THE PHASE 1 INPUT CONDITIONING CIRCUITRY

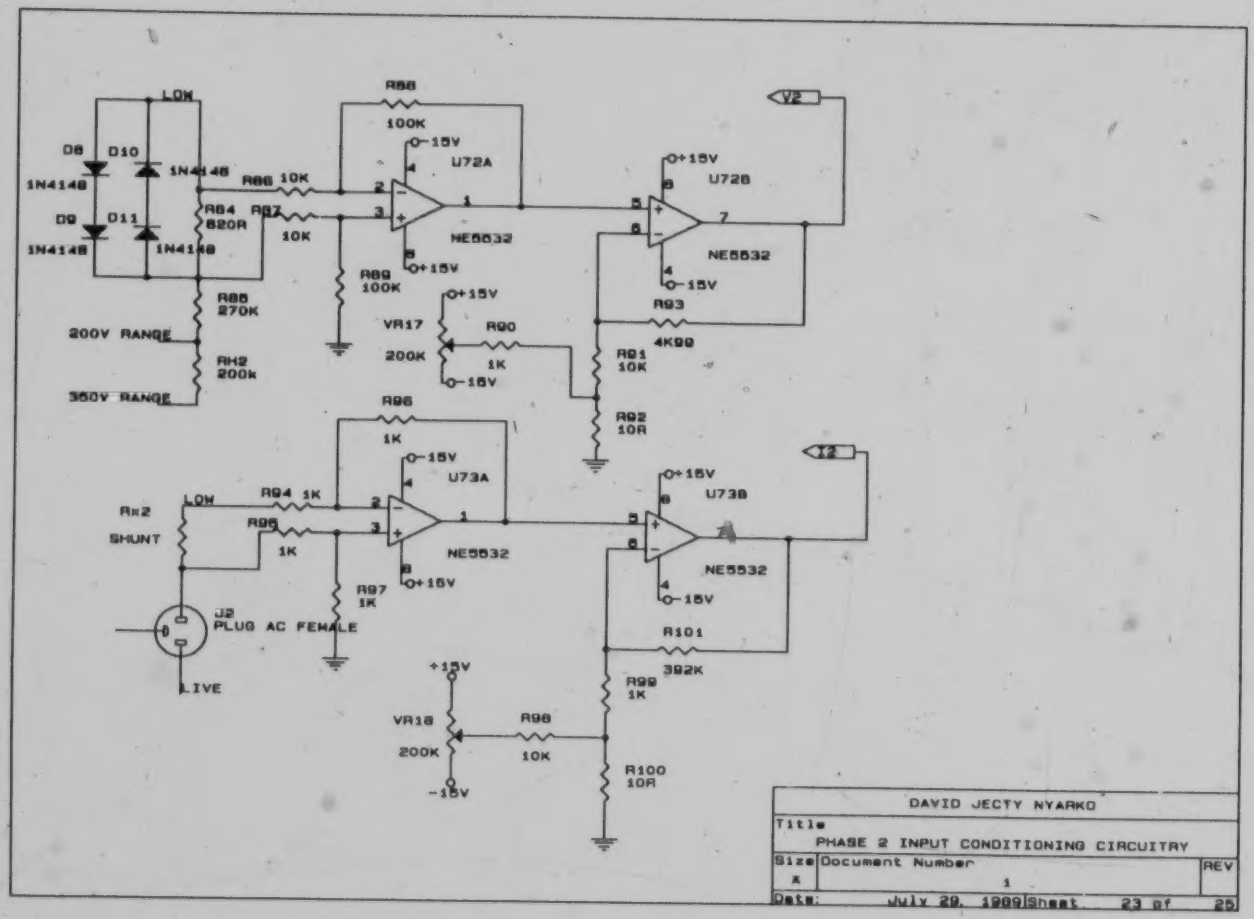


FIG. A.23 DETAILED SCHEMATIC OF THE PHASE 2 INPUT CONDITIONING CIRCUITRY

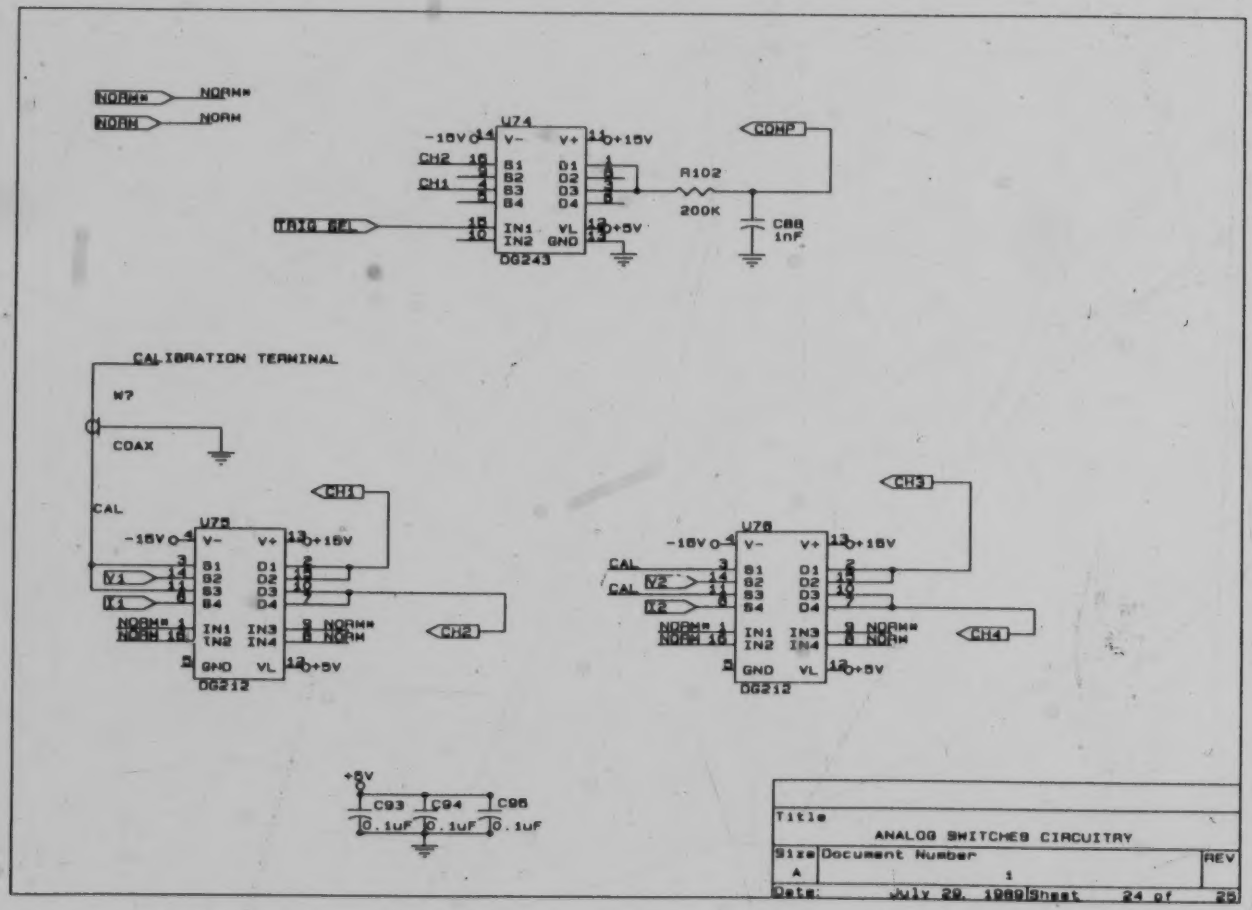


FIG. A.24 DETAILED SCHEMATIC OF THE ANALOG SWITCHES CIRCUITRY

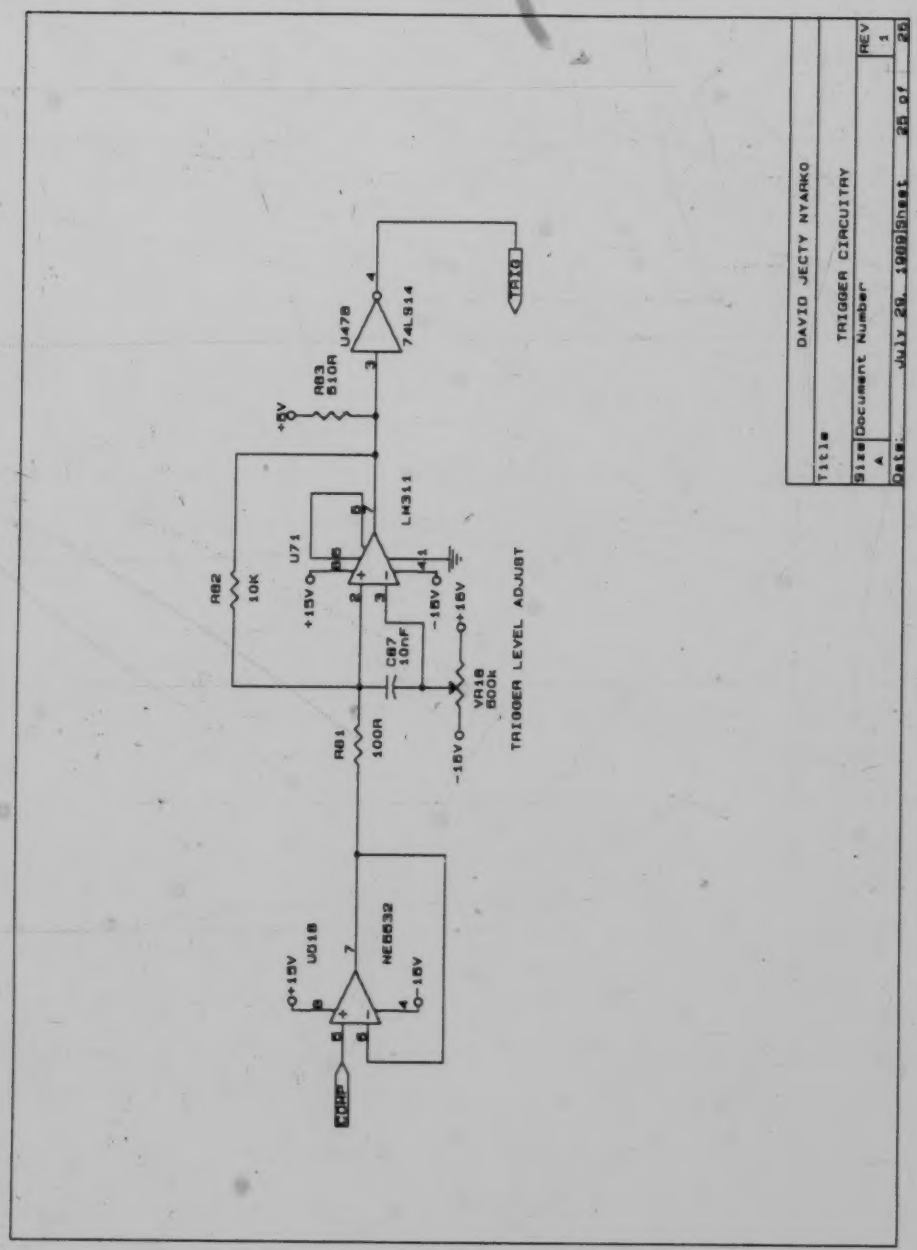


FIG. A.25 DETAILED SCHEMATIC OF THE TRIGGER CIRCUITRY

APPENDIX B SOFTWARE LISTING

```
;
;
        section  0
;
ramstart equ        $200000
ramend   equ        $20ffff
;
; lcd display registers
;
disp     equ        $6c0000        ;display
;
;  gpib registers
;
gpia     equ      $640000        ;gpia base register
instatr  equ      0              ;interrupt status  (r0r)
inmskr   equ      instatr        ;interrupt mask  (r0w)
cstatr   equ      2              ;command status (r1r)
adstatr  equ      4              ;address status (r2r)
admodr   equ      adstatr        ;address mode  (r2w)
auxcr    equ      6              ;auxiliary command (r3r)
adsw     equ      8              ;address switch (r4r)
adrg     equ      adsw           ;address register (r4w)
spolr    equ      10             ;serial poll (r5)
cpastr   equ      12             ;command pass through (r6)
ppolr    equ      cpastr         ;parallel poll (r6w)
dtinr    equ      14             ;data in (r7r)
dtoutr   equ      dtinr          ;data out (r7w)
;
; pia registers
;
piada    equ        $680000        ;pia base register
piadb    equ        piada+2        ;pia data reg. b
piacra   equ        piada+4        ;pia control reg. a
piacrb   equ        piada+6        ;pia control reg. b
;
; programmable keyboard/display
; interface register
;
keyb     equ        $700000
;
; ptm registers
;
time     equ      $600000
ctl13    equ      0          ;control register 1,3
ctl2     equ      2          ;control register 2
status   equ      2          ;status register
1th1     equ      4          ;timer 1 latch   (msb)
cnt1h    equ      4          ;timer 1 counter (msb)
1th2     equ      8          ;timer 2 latch   (msb)
cnt2h    equ      8          ;timer 2 counter (msb)
```

```
lth3     equ     12        ;timer 3 latch   (msb)
cnt3h    equ     12        ;timer 3 counter (msb)
;
; other equates
;
intsoff equ      $0f00
intson  equ      $f8ff
;
false_key equ 5
local   equ      %00010000
remote  equ      %11101111
key_dis equ      %11110111
key_en  equ      %00001000
stac_dis  equ    %11111001
stac_en equ      %00000110
ac_en    equ      %00000100
calib_on  equ    %01000000
calib_off equ    %10111111
norm_on equ      %00100000
norm_off  equ    %11011111
;
samp_st    equ   $1ffffe
;
line1    equ     0
line2    equ     $40
line3    equ     $14
line4    equ     $54
;
pt       equ     $a5
;
blink_on  equ    $f
blink_off equ    $e
;
adc1 equ         $400000
adc2 equ         $400002
adc3 equ         $400004
adc4 equ         $400006
;
; data backgrounds
;
dbgnd_1  equ  0
dbgnd_2  equ  $5555
dbgnd_3  equ  $3333
dbgnd_4  equ  $0f0f
dbgnd_5  equ  $00ff
;
;
tim_latch equ    $400009
;
stack_top equ    ramend-1     ;stack pointer
stack_base equ     stack_top-320
;
```

```
;   scale factors
;
ch1low_num     equ  2020
ch1high_num    equ  3638
ch2_num        equ  201
ch3low_num     equ  2020
ch3high_num    equ  3638
ch4_num        equ  201
;
ch1_denom      equ  11
ch2_denom      equ  10
ch3_denom      equ  11
ch4_denom      equ  10
w1_denom equ   17
w2_denom equ   17
;
;**************
; main program
;**************
;
         dc.l     stack_top     ;reset stack pointer
         dc.l     start      ;reset program counter
         dc.l     b_excpt    ;bus error
         dc.l     a_excpt    ;address error
         dc.l     i_excpt    ;illegal instruction
         dc.l     d_excpt    ;divide by zero
         dc.l     c_excpt    ;chk
         dc.l     o_excpt    ;trapv
         dc.l     p_excpt    ;privilege
i_trace  dc.l     t_excpt    ;trace
         dc.l     x_excpt    ;11010
         dc.l     y_excpt    ;11111
         ds.l     12         ;unassigned as yet
         dc.l     s_excpt    ;spurious iterrupt
;
;        autovectored interrupts
;
;
         dc.l     int        ;level 1
         dc.l     int        ;level 2
         dc.l     int        ;level 3
         dc.l     int_tim    ;level 4 autovector (timer)
         dc.l     int_key    ;level 5 autovector  (8279)
         dc.l     int_gpia   ;level 6 autovector (gpia)
         dc.l     int7       ;level 7
;
         ds.l     224
;
;**********************
; program start
;**********************
;
```

```
;
start:
        reset                           ;reset all devices
        ori.w   #intsoff,sr             ;disable interrupts
;
pia:
    lea     piada,a0
    moveq   #0,d0
    movep.w d0,4(a0)
    not.w   d0
    movep.w d0,0(a0)          ;ports a and b as outputs
    move.w  #$404,d0          ;point to dra and drb
    movep.w d0,4(a0)
    not.w     d0
    movep.w d0,0(a0)
key:
        lea       keyb,a0
        move.b    #%00000011,2(a0)
        move.b    #%00101000,2(a0)
        move.b    #%01000000,2(a0)
        move.b    #%11000001,2(a0)
timer:
    lea       time,a0
    move.w    #$ffff,d0
    movep.w   d0,1th1(a0)   ;initialize timer counter 1
    movep.w   d0,1th2(a0)   ;initialize timer counter 2
    movep.w   d0,1th3(a0)   ;initialize timer counter 3
;
    move.b  #%00001010,ctl13(a0) ;timer 3 external inputs
;                                  freq mode ints. disabled
lcd_display:
;
        lea     disp,a0
;
        moveq.l #-2,d0
        moveq.l #-2,d1
        moveq.l #-2,d2
        moveq.l #-2,d3
;
        move.b  #$30,(a0)     ;internal reset
lcd1    dbra    d0,lcd1
        move.b  #$30,(a0)
lcd2    dbra    d1,lcd2
        move.b  #$30,(a0)
lcd3    dbra    d2,lcd3
;
        move.b  #$38,(a0)    ;2 line mode 8
;                            bit data transfer
busy_1  dbra    d3,busy_1
;
        move.b  #$d,(a0)
busy_2  move.b  (a0),d0
```

```
        bmi.s   busy_2
;
        move.b  #6,(a0)         ;set entry mode
busy_3  move.b  (a0),d0
        bmi.s   busy_3
;
gpia_:
        lea     gpia,a1
;
        move.b  adsw(a1),d0     ;read address switch
        move.b  d0,adrg(a1)     ;write to address reg.
        move.b  #0,auxcr(a1)      ;clear gpib reset state
        move.b  #0,inmskr(a1)     ;clear interrupts
;
    bsr     clear
;
    move.b  #blink_on,(a0)      ;set the cursor blinking
    move.b  #blink_on,d6      ;save the state of the display
blon1   move.b  (a0),d0
    bmi.s   blon1
;
    moveq   #line2,d3
    lea     messg1(pc),a5
    bsr     display
;
; memory(13N) test for static rams
;
; a2 holds failing address
;
    moveq.l   #-14,d5
;
    lea.l     ramstart,a0
    lea.l     ramend-1,a1
;
    move.l    #dbgnd_1,d0
    move.b    #1,piada
memtest:
    move.w    d0,d1
    move.w    d0,d2
    not.w     d2
    clr.w     d3
    clr.w     d4
;
    move.l    a0,a2
;
mem_init:           ;Initialization Wr(0)
    move.w    d1,(a2)+
    cmpa.l    a2,a1
    bcc.s     mem_init
;
m1234:
    tst.w     d4
```

```
        beq.s     m12_init
        movea.l   a1,a2
        bra.s     m1234a
    ;
    m12_init:
        movea.l   a0,a2
    ;
    m1234a:
        cmp.w     (a2),d1
        bne.s     ramerr
        move.w    d2,(a2)
        cmp.w     (a2),d2
        bne.s     ramerr
        tst.w     d4
        beq.s     m1234b
        subq.l    #2,a2
        cmpa.l    a0,a2
        bra.s     m1234c
    m1234b:
        addq.l    #2,a2
        cmpa.l    a2,a1
    m1234c:
        bcc.s     m1234a
        exg.l     d1,d2
        not.w     d3
        bne.s     m1234
    ;
        not.w     d4
        bne.s     m1234
    ;
        cmpi.w    #42,d5
        beq  ram_ok    ;test successful
        addi.l    #14,d5
        jmp  dbg_table(pc,d5.w)
    ;
    dbg_table:
        move.w    #dbgnd_2,d0
        move.b    #2,piada
        bra.s     memtest
        move.w    #dbgnd_3,d0
        move.b    #4,piada
        bra.s     memtest
        move.w    #dbgnd_4,d0
        move.b    #8,piada
        bra.s     memtest
        move.w    #dbgnd_5,d0
        move.b    #16,piada
        bra  memtest
    ;
    ramerr  moveq   #line1,d3
            lea     messg2(pc),a5
            bsr     display
```

```
;
        lea.l   disp_buff1,a5
    move.l    #'A2= ',(a5)+
        move.l  a2,(a5)
        bsr     hex_ascii
    subq.l    #4,a5
        moveq   #line2,d3
        bsr     display
        lea.l   disp_buff2,a5
    move.l    #'D1= ',(a5)+
        move.l  d1,(a5)
    moveq.l   #2,d5
        bsr     hex_ascii
    subq.l    #4,a5
        moveq   #line3,d3
        bsr     display
;
        lea.l   disp_buff3,a5
    move.l    #'D2= ',(a5)+
        move.l  d2,(a5)
        bsr     hex_ascii
    subq.l    #4,a5
        moveq   #line4,d3       ;display failing
        bsr     display         ;address
;
here2   bra.s   here2
ram_ok:
    bclr.l    #0,d6
    move.b    d6,disp_state
    bsr  disp_ctrl
        move.b  #$ff,key_buff     ;initialize key buffer
        move.b  #0,pia_a
        move.b  pia_a,piada     ;turn off all leds
        move.b  #0,pia_b
        ori.b      #local,pia_b  ;keyboard on local control
;                  triger source = channel 1
        ori.b   #key_en,pia_b
        ori.b   #norm_on,pia_b
        andi.b  #calib_off,pia_b
    bclr.b    #7,pia_b  ;turn off remote led (no 8)
        move.b  pia_b,piadb     ;enable keyboard interrupts
        clr.b   prog_state      ;assume normal mode
    clr.b     state1
    move.w    #100,shunt_val1
    move.w    #100,shunt_val2
;
    move.w    #ch1low_num,tnumer_v1l
    move.w    #ch1high_num,tnumer_v1h
    move.w    #ch2_num,tnumer_i1
    move.w    #ch3low_num,tnumer_v2l
    move.w    #ch3high_num,tnumer_v2h
    move.w    #ch4_num,tnumer_i2
```

```
;
    move.w    #ch1_denom,denom_v1
    move.w    #ch2_denom,denom_i1
    move.w    #ch3_denom,denom_v2
    move.w    #ch4_denom,denom_i2
    move.w    #w1_denom,denom_w1
    move.w    #w2_denom,denom_w2
;
    bsr  watt_fact
    bsr  store_fact
;
; initialize  program mode variables
;
    move.l    #prog_buff,prog_max
    move.l    #prog_buff,prog_pointer
    clr.w     store_size
    move.l    #data_pointer+4,data_pointer
;
        move.b  #%11111111,gpia
        andi.w  #intson,sr      ;enable interrupts
;
    move.w    #$800,cal_buff1
    move.w    #$800,cal_buff2
    move.w    #$800,cal_buff3
    move.w    #$800,cal_buff4
;
    clr.b     menu_state
;
begin   clr.b   state
;
begin_0:
;
    lea.l     menu_state,a0
;
    tst.b     (a0)
    bne.s     begin_1
    bsr  menu0
    bra.s     wait
begin_1:
    cmpi.b    #1,(a0)
    bne.s     begin_2
    bsr  menu1
    bra.s     wait
begin_2:
    cmpi.b    #2,(a0)
    bne.s     begin_3
    bsr  menu2
    bra.s     wait
begin_3:
    cmpi.b    #3,(a0)
    bne.s     begin_4
    bsr  menu3
```

```
    bra.s     wait
begin_4:
    cmpi.b    #4,(a0)
    bne.s     begin_5
    bsr  menu4
    bra.s     wait
begin_5:
    cmpi.b    #5,(a0)
    bne.s     begin_6
    bsr  menu5
    bra.s     wait
begin_6:
    cmpi.b    #6,(a0)
    bne.s     begin_7
    bsr  menu6
    bra.s     wait
begin_7:
    cmpi.b    #7,(a0)
    bne.s     begin_0
    bsr  menu7
wait    andi.b  #%00110000,state
        tst.b   prog_state
        beq.s   wait_norm
    bsr  program
wait_norm:
        lea     key_buff,a0
        tst.b   (a0)          ;0 to start
        beq     start
    cmpi.b    #$12,(a0) ;a for recal (dc only)
    bne.s     waitb
    bset.b    #7,state
    bra.s     wait_3
waitb    cmpi.b  #$13,(a0)       ;b to sample
        beq     cont
    cmpi.b    #$11,(a0) ;9 to calibrate and compute
    bne.s     waitc           ;rms value
    bset.b    #6,state
    bra.s     wait_2
waitc    cmpi.b  #$18,(a0)       ;c to calibrate
        bne.s   waitd
wait_2  ori.b   #calib_on,pia_b
        andi.b  #norm_off,pia_b
        move.b  pia_b,piadb
wait_3   bsr     cal_display
waita0   btst.b    #0,state1
    beq.s     waita1
    move.b    auto_chan,(a0)
    bra.s     waita2
waita1  tst.b   prog_state
        beq.s   waita2
    bsr  program
waita2   tst.b   (a0)
```

```
          beq.s   waita2
          cmpi.b  #8,(a0)
          bne.s   waita3
          move.b  #4,(a0)
waita3    cmpi.b  #4,(a0)
          bhi.s   waita2
          move.b  (a0),cal_chan
          bra   cont
waitd     cmpi.b  #$19,(a0)       ;d for no. of acquisitions
          bne.s   waite
     bsr  clear
     moveq     #line3,d3
     lea  messg24(pc),a5 ;display ' SAMPLE SIZE'
     bsr  display
          bsr     acq_disp
          lea     acq_count,a1
          bsr     acq
          move.w  (a1),acq_count
          bra     begin_0
waite     cmpi.b  #$1a,(a0)       ;e for delay
          bne.s   wait1
     bsr  clear
     moveq     #line3,d3
     lea  messg25(pc),a5 ;display ' DELAY'
     bsr  display
          bsr     acq_disp
          lea     delay_value,a1
          bsr     acq
          move.w  (a1),delay_count
          bra     begin_0
wait1     cmpi.b    #1,(a0)    ;1 for menu display on/off
     bne.s      wait4
     bchg.b    #0,pia_a
     move.b    pia_a,piada     ;turn led 0 on/off
     bchg.b    #2,disp_state
     bsr  disp_ctrl
     move.b    #$ff,(a0) ;force return to menu
;
wait4     cmpi.b    #8,(a0)    ;4 for overload test on/off
     bne.s      wait7
     bchg.b    #2,state1
     bchg.b    #1,pia_a
     move.b    pia_a,piada
     move.b    #$ff,(a0)
wait7     cmpi.b    #$b,(a0)   ;7 pressed
     bne.s     wait2
     bchg.b    #2,pia_a
     bchg.b    #0,pia_b
     move.b    pia_a,piada
     move.b    pia_b,piadb
     move.b    #$ff,(a0)
;
```

```
wait2    cmpi.b    #2,(a0)          ;channel 1 high/low
    bne.s     wait3
    bchg.b    #3,pia_a
    move.b    pia_a,piada
    btst.b    #3,pia_a
    beq.s     wait2a
    bset.b    #6,state1 ;high range
    bra.s     wait2b
wait2a   bclr.b    #6,state1 ;low range
;
wait2b   bsr  store_fact
    move.b    #$ff,(a0) ;force return to menu
;
wait3    cmpi.b    #3,(a0)    ;channel 3 high/low
    bne.s     wait5
    bchg.b    #4,pia_a
    move.b    pia_a,piada
    btst.b    #4,pia_a
    beq.s     wait3a
    bset.b    #7,state1
    bra.s     wait3b
wait3a   bclr.b    #7,state1
wait3b   bsr  store_fact
    move.b    #$ff,(a0)
    bra  begin_0
wait5    cmpi.b    #9,(a0)    ;5 for scale factor numerators
    bne.s     wait6
    move.b    #$ff,(a0)
    move.w    #'N0',d2
    lea  numer_v1-4,a1
    bsr  factor
    bra  begin_0
;
wait6  cmpi.b #$a,(a0)  ;6 for scale factor denominators
    bne.s     waitc_0
    move.b    #$ff,(a0)
    move.w    #'D0',d2
    lea  denom_v1-4,a1
    bsr  factor
    bra  begin_0
waitc_0:
    cmpi.b    #$80,(a0) ;^0 for entering shunt values
    bne.s     waitc_1
    move.b    #$ff,(a0)
    bsr  shunt
    bra  begin_0         ;return to main menu
waitc_1:
    cmpi.b    #$81,(a0) ;^1 for lines remaining
    bne.s     waitc_2
    bsr  lines_left
    move.b    #$ff,(a0)
    bra  begin_0
```

```
waitc_2:
    cmpi.b    #$82,(a0) ;^2 display data
    bne.s     waitc_3
    bsr  disp_data
    bra  begin_0        ;return to main menu
waitc_3:
    cmpi.b    #$83,(a0) ;^3 storage on/off
    bne.s     waitc_4
    move.b    #$ff,(a0)
    bchg.b    #5,pia_a
    move.b    pia_a,piada
    btst.b    #5,pia_a
    beq.s     waitc_3a
    bset.b    #3,state1
    bra.s     waitc_4
waitc_3a:
    bclr.b    #3,state1
waitc_4:
    cmpi.b    #$88,(a0) ;^4 for keypad program
    bne.s     waitc_7
    bsr  automode
    bra  begin_0
waitc_7:
    cmpi.b    #$8b,(a0) ;^7 change scale factors
    bne.s     waitc_8
    bsr  sc_fact
    bra  begin_0
waitc_8:
    cmpi.b    #$90,(a0) ;a to perform 2nd calibration
;                   on all channels (dc only)
    bne.s     waitc_8a
    move.l    #$01020308,auto_chan
    bra.s     waitc_8b
waitc_8a:
    cmpi.b    #$9c,(a0)
    bne.s     wait8
waitc_8b:
    bset.b    #7,state
    bset.b    #0,state1
    bset.b    #0,state
    clr.w     recal_point
    bra  waita0
wait8   cmpi.b    #$10,(a0) ;8 to clear data
    bne.s     waitf
    clr.w     store_size
    move.l    #data_pointer+4,data_pointer  ;initialize
data pointer
    move.b    #$ff,(a0)
waitf   cmpi.b  #$1b,(a0)       ;f for frequency
    bne.s     waitc_f
    move.b    #14,meas_state ;in frequency mode
        bra     freq
```

```
waitc_f  cmpi.b    #$9b,(a0) ;^F next menu
    bne.s     waitc_b
    addq.b    #1,menu_state
    bclr.b    #3,menu_state
    move.b    #false_key,(a0)
    bra  begin_0
waitc_b
    cmpi.b    #$93,(a0) ;^B previous menu
    bne  wait
    move.b    #false_key,(a0)
    subq.b    #1,menu_state
    bpl.s     waitend
    andi.b    #%00000111,menu_state
waitend:
    bra  begin_0
cont     btst.b  #0,state
         bne.s   sample_0
         tst.b   prog_state
         beq.s   cont0
;
    bsr  program
;
cont0     bsr     clear
         moveq   #line1,d3
         lea     messg17(pc),a5
         bsr     display
         moveq   #line2,d3
         lea     messg18(pc),a5
         bsr     display
         moveq   #line3,d3
         lea     messg19(pc),a5
         bsr     display
         moveq   #line4,d3
         lea     messg19a(pc),a5
         bsr     display
waitc0    cmpi.b  #1,(a0)
         bne.s   waitc1
         bset.b  #1,state
         bra.s   sample_0
waitc1   cmpi.b  #2,(a0)
         bne.s   waitc2
         bset.b  #2,state
         bra.s   sample_0
waitc2   cmpi.b  #3,(a0)
         bne.s   waitc0
         bset.b  #3,state
sample_0:
         bsr     clear
         move.b  #$ff,key_buff
sample:
         bsr     init_samp
    bsr  init_sampac
```

```
        btst.b  #0,state
        beq.s   sample_a
    btst.b    #6,state  ;if rms value is
    bne.s     sample_a  required, continue
    btst.b    #7,state
    bne.s     sample_a  ;if in recal mode,skip
        clr.l   d2
        clr.l   d3
        clr.l   d4
        clr.l   d5
        bra.s   samp0
sample_a:
        move.w  cal_buff1,d2
        move.w  cal_buff2,d3
        move.w  cal_buff3,d4
        move.w  cal_buff4,d5
samp0   addq.l  #8,a5
samp1   tst.b   (a6)          ;if low freq, use dc mode
        beq     dc_samp       ;else, use ac mode
        btst.b  #0,(a0)       ;data stored in latches?
        beq.s   samp1
        move.w  (a4),(a5)
        move.w  (a3),-(a5)
        move.w  (a2),-(a5)
        move.w  (a1),-(a5)
        sub.w   d2,(a5)+
        sub.w   d3,(a5)+
        sub.w   d4,(a5)+
        sub.w   d5,(a5)
        move.b  (a0),d1
    move.b    d1,d7
    and.w     d0,d1
        jmp     zc_tab(pc,d1.w)
zc_tab:
        bra.s   sample
        bra.s   sample
        bra.s   sample
        bra.s   sample
        bra.s   stg_1a  ;1 more sample required
        bra.s   samp0    ;just started sampling,continue
        bra.s   stg_2a  ;2 more samples required
        bra.s   samp0   ;continue sampling
stg_2a  addq.l  #8,a5
stg_2   btst.b  #0,(a0)
        beq.s   stg_2
        move.w  (a4),(a5)
        move.w  (a3),-(a5)
        move.w  (a2),-(a5)
        move.w  (a1),-(a5)
        sub.w   d2,(a5)+
        sub.w   d3,(a5)+
        sub.w   d4,(a5)+
```

```
          sub.w   d5,(a5)
stg_1a    addq.l  #8,a5
stg_1     btst.b  #0,(a0)
          beq.s   stg_1
          move.w  (a4),(a5)
          move.w  (a3),-(a5)
          move.w  (a2),-(a5)
          move.w  (a1),-(a5)
;
          sub.w   d2,(a5)+
          sub.w   d3,(a5)+
          sub.w   d4,(a5)+
          sub.w   d5,(a5)
;
;
    andi.b  #stac_dis,pia_b  ;start' and ac'(to clear ffs)
from pia
    ori.b   #key_en,pia_b  ;enable keyboard interrupts
    move.b  pia_b,piadb
;
    asr.b     #4,d7
    bmi.s     stg_3
    subq.l    #8,a5          ;adjust a5
;
stg_3    ext.w     d7
    move.w    d7,alpha16_store    ;store 16 * delta
    btst.b    #2,state1
    bne.s     sampla          ;neglect overload test
    bsr  ov_load
    btst.b    #1,state1
    beq.s     sampla          ;no overload
    bset.b    #0,disp_state
    bsr  disp_ctrl ;set cursor blinking
    bra  samp4bla
sampla   bclr.b    #0,disp_state
    bsr  disp_ctrl ;set blinking cursor off
    bsr  clear_regs
         move.l  a5,samp_end
         move.l  a5,d5
         sub.l   #$20000e,d5
         asr.l   #3,d5
         asl.w   #4,d5    ;multiply by 16 (16n)
         lea     $200008,a1  ;point to start of samples (Y0
chan 1)
    bsr  clear_regs
         btst.b  #0,state
         beq.s   samp3
    btst.b    #6,state
    bne.s     samp2b
      bsr     ac_calibrate
    move.b    #0,meas_state  ;in ac_calibrate state
    bsr  sign
```

```
samp2a  lea     disp_buff,a5
    move.w    #'R0',d1
    add.b     cal_chan,d1
    move.w    d1,(a5)+
    bsr  update
        bra     samp4
samp2b   bsr  ac_rms
    move.b    #4,meas_state  ;in ac_rms state
    bra.s     samp2a
samp3    btst.b    #3,state  ;3-phase mode selected?
    beq.s     samp3_0
    move.b    #10,meas_state ;in 3-phase ac mode
    bsr  phase3          ;compute power of both phases
    bsr  pf3_100         ;calculate 100 * pf(power factor)
    lea  disp_buff,a5
    move.l    #'PF =',(a5)+
    move.w    pf_store,d0
    move.w    #'  ',(a5)+
    move.w    Vmax,d1
    cmp.w     Imax,d1
    bcc.s     lead0
    move.w    #' -',(a5)+    ;lagging power factor
    bra  samp3_2
lead0    move.w    #' +',(a5)+    ;leading power factor
samp3_2:
    move.w    d0,(a5)
    moveq.l   #2,d5          ;two bytes
    bsr  hex_ascii ;convert to ascii
    move.b    1(a5),d0
    move.b    #'.',1(a5)     ;fix decimal point
    move.b    d0,(a5)
    bra  samp4
samp3_0:
    btst.b  #1,state
        beq.s   samp3a
        addq.l  #8,a1   ;point to Y1 chan1
        movea.l a1,a4
        movea.l a5,a2
        movea.l a5,a3
;
; for Vrms
;
        subq.l  #6,a2
        subq.l  #6,a3
        bra.s   samp3b
samp3a  btst.b  #2,state
        bne.s   samp3a_0
samp3a_0:
        adda.l  #12,a1    ;point to Y1( chan 3)
        movea.l a1,a4
        movea.l a5,a2
        movea.l a5,a3
```

```
        subq.l  #2,a2
        subq.l  #2,a3
samp3b  move.b    #6,meas_state  ;in normal ac mode
    move.l    a0,-(sp)
    lea  Vmax,a0
    bsr  max_sample
    move.l    (sp)+,a0
    bsr     correct
    bsr     evaluate
        move.l  4(a0),d0
        bsr     sqrt
    movem.l  d1/d2/a5/a6,-(sp)
;
    clr.l     d2
    btst.b    #1,state
    beq.s     fac_v1
    move.w    denom_v1,d2
    muls.w    numer_v1,d0
    bra.s     fac_v3
;
fac_v1  btst.b    #2,state
    bne.s     fac_v2
    movea.l   #6,a2           ;TEST
    bra  ramerr
fac_v2  move.w    denom_v2,d2
    muls.w    numer_v2,d0
fac_v3  asr.l     d2,d0     ;for 10 * result( to ldp)
    bcc.s     fac_v4
fac_v4  move.l    d0,d1
    clr.l     d0        ;prepare for routine
    bsr  bcd1       ;convert to bcd
        lea.l   disp_buff1,a5
        move.b  #'V',(a5)+
        bsr     update
    bsr  blank
    move.b    (a6),2(a6)      ;move termination character
    move.b    #'V',1(a6)
    move.b    -1(a6),(a6)
    move.b    #'.',-1(a6)     ;insert decimal point
;
        movem.l  (sp)+,d1/d2/a5/a6     ;restore d1/d2/a5/a6
;
; I rms
;
        addq.l  #2,a1         ;initialize to Y1
        addq.l  #2,a2         ;initialize to Yn
        addq.l  #2,a3         ;initialize to Yn
        addq.l  #2,a4         ;initialize to Y1
    bsr  clear_regs
    move.l    a0,-(sp)
    lea  Imax,a0
    bsr  max_sample
```

```
        move.l    (sp)+,a0
        bsr     correct
            bsr     evaluate
            move.l  4(a0),d0
        move.l    d0,Imsqr
            bsr     sqrt
        movem.l  d1/d2/a5/a6,-(sp)
        clr.l     d2
        btst.b    #1,state
        beq.s     fac_i1
        move.w    denom_i1,d2
        muls.w    numer_i1,d0
        bra.s     fac_i3
fac_i1  btst.b    #2,state
        bne.s     fac_i2
        movea.l   #7,a2           ;TEST
        bra  ramerr
fac_i2  move.w    denom_i2,d2
        muls.w    numer_i2,d0
fac_i3  asr.l     d2,d0     ;for 100 * result (to 2 dp)
        bcc.s     fac_i4
        addq.l    #1,d0           ;round up
fac_i4  move.l    d0,d1
        clr.l     d0        ;prepare for routine
        bsr  bcd1       ;convert to bcd
            lea.l   disp_buff2,a5
            move.b  #'I',(a5)+
            bsr     update
        bsr  blank
        move.b    (a6),2(a6)      ;move termination character
        move.b    #'A',1(a6)
        move.b    -1(a6),(a6)
        move.b    -2(a6),-1(a6)
        move.b    #'.',-2(a6)     ;insert decimal point
            movem.l  (sp)+,d1/d2/a5/a6      ;restore d1/d2/a5
;
;  W (power)
;
            subq.l  #2,a1
            subq.l  #2,a2
        bsr  clear_regs
            bsr     correct
            bsr     evaluate
            move.l  4(a0),d0        ;save a5
        move.l    d0,Watt
        bsr  watt_bcd
        move.l    a5,-(sp)
            lea.l   disp_buff3,a5
            move.b  #'W',(a5)+
            bsr     update
        bsr  blank
        move.b    (a6),2(a6)
```

```
        move.b    #'W',1(a6)
        move.b    -1(a6),(a6)
        move.b    #'.',-1(a6)
        move.l    (sp)+,a5
    ;
        bsr  pf_100          ;calculate 100 * pf(power factor)
    ;
        lea  disp_buff,a5
        move.l    #'PF =',(a5)+
        move.w    pf_store,d0
        move.w    #'  ',(a5)+
        move.w    Vmax,d1
        cmp.w     Imax,d1
        bcc.s     lead
        move.w    #' -',(a5)+    ;lagging power factor
        bra.s     samp3c_0
    lead    move.w    #' +',(a5)+    ;leading power factor
    ;
        move.w    d0,(a5)
        moveq.l   #4,d5
        bsr  hex_ascii
        move.b    1(a5),d0
        move.b    #'.',1(a5)
        move.b    d0,(a5)
    ;
    samp4    btst.b    #5,state
        beq.s     samp4_0
        subq.w    #1,delay_count
        bne  samp4b1a
    ;
    samp4_0 move.w  delay_value,delay_count
            moveq   #line1,d3
            btst.b  #0,state
            beq.s   samp4a
            lea     messg9(pc),a5
            bsr     display
            moveq   #line3,d3
            lea     disp_buff,a5
            bsr     display
    ;
            bra.s samp4a4
    samp4a   moveq     #line1,d3
        lea  disp_buff,a5   ;display PF(power factor)
        bsr  display
        moveq   #line2,d3
        lea     disp_buff1,a5   ;display VOLTAGE/POWER 1
        bsr     display
    ;
        moveq   #line3,d3
        lea     disp_buff2,a5   ;display CURRENT/POWER 2
        bsr     display
       moveq     #line4,d3
```

```
    btst.b    #3,state
    beq.s     samp4a2
    lea       messg26(pc),a5
    bra.s     samp4a3
samp4a2:
    lea       disp_buff3,a5   ;display POWER
samp4a3:
    bsr       display
samp4a4:
    btst.b    #3,state1
    beq.s     samp4b
;
    bsr  store_data
;
samp4b  btst.b  #4,state
        beq.s   samp4b1a
    subq.w    #1,acq_count
    beq.s     samp4b2
samp4b1a:
    cmpi.b  #$ff,key_buff
        beq     sample
samp4b2 btst.b  #0,state
        beq     begin_0
        ori.b   #norm_on,pia_b
        andi.b  #calib_off,pia_b
        move.b  pia_b,piadb
        bra     begin_0
;
dc_samp:
        bsr     init_samp
;
        andi.b  #stac_dis,pia_b ;start' and ac' from pia
        andi.b  #key_dis,pia_b
        move.b  pia_b,piadb
        move.l  #1025,d6
        btst.b  #0,state
        beq.s   dc_sampa
        btst.b  #7,state
        bne.s   dc_sampa     ;skip if in recal mode
        clr.l   d2
        clr.l   d3
        clr.l   d4
        clr.l   d5
        bra.s   dc_samp0
dc_sampa:
        move.w  cal_buff1,d2
        move.w  cal_buff2,d3
        move.w  cal_buff3,d4
        move.w  cal_buff4,d5
;
dc_samp0:
        addq.l  #8,a5
```

```
dc_samp1:
        btst.b  #0,(a0)
        beq.s   dc_samp1
        move.w  (a4),(a5)
        move.w  (a3),-(a5)
        move.w  (a2),-(a5)
        move.w  (a1),-(a5)
        sub.w   d2,(a5)+
        sub.w   d3,(a5)+
        sub.w   d4,(a5)+
        sub.w   d5,(a5)
;
        dbra    d7,dc_samp0
;
        ori.b   #key_en,pia_b  ;enable keyboard interrupts
        move.b  pia_b,piadb
;
    btst.b    #2,state1
    bne.s     dc_samp0a ;neglect overload test
    bsr  ov_load
    btst.b    #1,state1
    beq.s     dc_samp0a
    bset.b    #0,disp_state
    bsr  disp_ctrl ;set cursor blinking
    bra  dc_samp2b1
;
dc_samp0a:
    bclr.b    #0,disp_state
    bsr  disp_ctrl ;set cursor blinking
    bsr  clear_regs
    move.l  a5,samp_end
    lea     $200008,a1  ;point to start of samples( Y0)
;                                (channel 1)
    btst.b  #0,state
    beq.s   dc_samp1a
    move.b    #2,meas_state  ;in calibrate dc mode
        bsr     dc_calib
        bsr     sign
        lea     disp_buff,a5
    move.w    #'R0',d1
    add.b     cal_chan,d1
    move.w    d1,(a5)+
        bsr     update
        bra        dc_samp1c
dc_samp1a:
    btst.b    #4,state
    beq  dc_samp1a0
    adda.l    #12,a1          ;point to Y1 chan 3
    movea.l   a5,a2
    movea.l   a5,a3           ;point to Yn chan 4
    subq.l    #2,a2           ;point to Yn chan 3
;
```

```
        move.b    #12,meas_state ;in 3-phase dc mode
        bsr  dc_power
        move.l    d0,power2
        bset.b    #2,state  ;force a use of watt_bcd routine
        bsr  watt_bcd
        bclr.b    #2,state
        move.l    a5,-(sp)  ;prepare to display
            lea.l   disp_buff2,a5
        move.l    #'W2= ',(a5)+
            bsr     update
        bsr  blank
        move.b    (a6),2(a6)
        move.b    #'W',1(a6)
        move.b    -1(a6),(a6)
        move.b    #'.',-1(a6)
        move.l    (sp)+,a5
        subq.l    #4,a1
        subq.l    #4,a5
        movea.l   a5,a2
        movea.l   a5,a3
        subq.l    #4,a2
;
        bsr  dc_power
;
        move.l    d0,power1
        bsr  watt_bcd
        move.l    a5,-(sp)  ;prepare to display
        lea.l   disp_buff1,a5
        move.l    #'W1= ',(a5)+
            bsr     update
        bsr  blank
        move.b    (a6),2(a6)
        move.b    #'W',1(a6)
        move.b    -1(a6),(a6)
        move.b    #'.',-1(a6)
        move.l    (sp)+,a5
        bra  dc_sample
;
dc_sample0:
            btst.b  #1,state
            beq.s   dc_sample1
;
            subq.l  #6,a5
            bra.s   dc_sample3
dc_sample1:
            btst.b  #2,state
            bne.s   dc_sample2
dc_sample2:
            addq.l  #4,a1           ;Y0 (channel 3)
            subq.l  #2,a5           ;Yn (channel 3)
;
; V avrg.
```

```
;
dc_sampla3:
    move.b    #8,meas_state  ;in normal dc mode
     bsr      dc_avrg
        bsr      sign
        movem.l  d1/d2/a5/a6,-(sp)     ;save d1/d2/a5/a6
    clr.l     d2
    btst.b    #1,state
    beq.s     fdc_v1
    move.w    denom_v1,d2
    muls.w    numer_v1,d0
    bra.s     fdc_v3
fdc_v1   btst.b    #2,state
fdc_v2   move.w    denom_v2,d2
    muls.w    numer_v2,d0
fdc_v3   asr.l     d2,d0
    bcc.s     fdc_v4
    addq.l    #1,d0
fdc_v4   move.l    d0,d1
;
    clr.l     d0
    bsr  bcd1
        lea.l   disp_buff1,a5
        move.b  #'V',(a5)+
        bsr     update
;
    bsr  blank
;
    move.b    (a6),2(a6)
    move.b    #'V',1(a6)
    move.b    -1(a6),(a6)
    move.b    #',',-1(a6)
    movem.l   (sp)+,d1/d2/a5/a6     ;restore d1/d2/a5/a6
;
; I avrg
;
        addq.l  #2,a1
        addq.l  #2,a5
    bsr  clear_regs
        bsr     dc_avrg
        bsr     sign
        movem.l  d1/d2/a5/a6,-(sp)     ;save d1/d2/a5/a6
    clr.l     d2
    btst.b    #1,state
    beq.s     fdc_i1
    move.w    denom_i1,d2
    muls.w    numer_i1,d0
    bra.s     fdc_i3
fdc_i1   btst.b    #2,state
    bne.s     fdc_i2
    movea.l   #7,a2
    bra  ramerr
```

```
fdc_i2   move.w    denom_i2,d2
    muls.w    numer_i2,d0
fdc_i3   asr.l     d2,d0
    bcc.s     fdc_i4
    addq.l    #1,d0
fdc_i4   move.l    d0,d1
    clr.l     d0
    bsr  bcd1
        lea.l   disp_buff2,a5
        move.b  #'I',(a5)+
        bsr     update
;
    bsr  blank
;
    move.b    (a6),2(a6)
    move.b    #'A',1(a6)
    move.b    -1(a6),(a6)
    move.b    #',',-2(a6)
    movem.l   (sp)+,d1/d2/a5/a6      ;restore d1/d2/a5/a6
;
; W (power)
;
        movea.l a5,a2
        movea.l a5,a3
        subq.l  #2,a2
        addq.l  #6,a1
;
    bsr  clear_regs
;
    bsr  dc_power
    bsr  watt_bcd
    move.l    a5,-(sp)
        lea.l   disp_buff3,a5
        move.b  #'W',(a5)+
        bsr     update
    bsr  blank
    move.b    (a6),2(a6)
    move.b    #'W',1(a6)
    move.b    -1(a6),(a6)
    move.b    #'.',-1(a6)
;
    move.l  (sp)+,a5          ;restore a5
;
dc_samp1c:
    btst.b    #5,state
    beq.s     dc_samp1c0
    subq.w    #1,delay_count
    bne  dc_samp2b1
dc_samp1c0:
    move.w  delay_value,delay_count
        moveq   #line1,d3
        btst.b  #0,state
```

```
        beq.s   dc_samp1d
        lea     messg10(pc),a5
        bsr     display
        moveq   #line3,d3
        lea     disp_buff,a5
        bsr     display
        bra.s dc_samp1d2
dc_samp1d:
        lea     messg10a(pc),a5
        bsr     display
        moveq   #line2,d3
        lea     disp_buff1,a5   ;display VOLTAGE/POWER1
        bsr     display
        moveq   #line3,d3
        lea     disp_buff2,a5   ;display CURRENT/POWER2
        bsr     display
    moveq     #line4,d3
    btst.b    #3,state
    beq.s     dc_samp1d0
    lea     messg26(pc),a5
    bra.s     dc_samp1d1
dc_samp1d0:
        lea     disp_buff3,a5   ;display POWER
dc_samp1d1:
        bsr     display
;
dc_samp1d2:
    btst.b    #4,state1
    beq.s     dc_samp1f
;
    bsr  store_data
dc_samp1f:
    btst.b  #4,state
    beq.s   dc_samp2b1
    subq.w    #1,acq_count
    beq.s     dc_samp2c
;
dc_samp2b1:
        cmpi.b  #$ff,key_buff
        beq     sample
dc_samp2c:
        btst.b  #0,state
        beq     begin_0
;
        ori.b   #norm_off,pia_b
        andi.b  #calib_off,pia_b
        move.b  pia_b,piadb
;
        bra     begin_0
;
;
;  frequency
```

```
;
freq    bsr     clear
        move.w  #$ffff,freq_max
        clr.w   freq_min
        moveq   #line1,d3
        lea     messg8(pc),a5
        bsr     display
        moveq   #line2,d3
        lea     messg8a(pc),a5
        bsr     display
        moveq   #line3,d3
        lea     messg8b(pc),a5
        bsr     display
        moveq   #line4,d3
        lea     messg8c(pc),a5
        bsr     display
;
        move.b  #$ff,key_buff
        move.b  #%10100011,time+2
        move.b  #%10000010,time
;
freq0   lea     time,a5
        moveq   #-1,d6
;
freq1   subq    #1,d6
        beq     freq3
        btst.b  #2,status(a5)
        beq.s   freq1
        movep.w cnt3h(a5),d0
        not.w   d0
        clr.b   freq_state
    btst.b    #5,state
    beq.s     freq1a
    cmpi.w    #1,delay_count
    bhi.s     freq1c
freq1a  cmp.w   freq_min,d0
        bls.s   freq1b
        bset.b  #0,freq_state   ;set new min value flag
        move.w  d0,freq_min     ;higher count => lower freq
freq1b  cmp.w   freq_max,d0
        bcc.s   freq1c
        bset.b  #1,freq_state   ;set new max value flag
        move.w  d0,freq_max     ;lower count => higher freq
freq1c  clr.l d2
        move.w  d0,d2
;
        moveq.l  #10,d4
;
        move.l  #800000,d1
        divu    d0,d1
    move.l    d1,d3
    clr.l     d1
```

```
        move.w    d3,d1
;
            clr.l d0
            bsr     bcd1
            move.w  d0,disp_buff
            lea     disp_buff,a5
            moveq   #2,d5
            bsr     hex_ascii
            swap    d3
            mulu    d4,d3
            divu    d2,d3
            clr.l d0
        clr.l     d1
            move.w  d3,d1
            bsr     bcd1
            move.b  #pt,(a6)
            addq.l  #1,a6
            movea.l a6,a5
            move.b  d0,(a5)
            moveq   #1,d5
            bsr     hex_ascii       ;convert to ascii
            cmpi.b  #'0',disp_buff
            bne.s   freq2
            move.b  #' ',disp_buff
            cmpi.b  #'0',disp_buff+1
            bne.s   freq2
            move.b  #' ',disp_buff+1
freq2     btst.b    #5,state
        beq.s     freq2a
        subq.w    #1,delay_count
        bne   freq6
freq2a    move.w    delay_value,delay_count
        lea     disp_buff,a5
        move.l    (a5)+,disp_buff1
        move.l    (a5),disp_buff1+4
        lea   disp_buff1,a5
            moveq   #line2,d3
            bsr     display
            btst.b  #0,freq_state
            beq.s   freq2b
            lea     disp_buff,a5
        move.l    (a5)+,disp_buff2+2
        move.l    (a5),disp_buff2+6
        lea   disp_buff2,a5
            moveq   #line3,d3       ;display new min freq
            bsr     display
freq2b    btst.b  #1,freq_state
            beq.s   freq3
            lea     disp_buff,a5
        move.l    (a5)+,disp_buff3+2
        move.l    (a5),disp_buff3+6
        lea   disp_buff3,a5
```

```
        moveq   #line4,d3       ;display new max freq
        bsr     display
freq3   tst.b   d6
        bne.s   freq4
        moveq.l #line2,d3
        lea     messg12(pc),a5
        bsr     display
    move.l    a4,-(sp)
        lea     disp_buff,a4
freq3a   move.b    (a5)+,(a4)+
    bne.s     freq3a
    move.l    (sp)+,a4
freq4    btst.b    #3,state1
    beq.s     freq5
    bsr  store_data
freq5    btst.b    #3,state
    beq.s     freq6
;
    subq.w    #1,acq_count
    beq  begin_0
freq6   cmpi.b  #$ff,key_buff
        bne     begin_0
;
        bra     freq0
;
;
;*****************************
;                            *
;  subroutines               *
;                            *
;*****************************
;
;
;*****************************
; subroutine disp_ctrl
;
; purpose: to turn display,cursor and blink
;      of cursor on/off
;
;*****************************
disp_ctrl:
    movem.l   d0/a0,-(sp)
        lea     disp,a0
        move.b  disp_state,(a0)
disp_ctrl1:
    move.b  (a0),d0          ;set cursor blinking off
       bmi.s   disp_ctrl1
    movem.l   (sp)+,d0/a0
    rts
;
;
;
```

```
; subroutine menu0
; purpose: to display 0th menu
;*****************************
;
menu0:
    bsr  clear
    moveq     #line1,d3
    lea  messg60(pc),a5 ;METER READY
    bsr  display
    moveq     #line3,d3
    lea  messg38(pc),a5 ;^F: NEXT MENU
    bsr  display
    moveq     #line4,d3
    lea  messg39(pc),a5 ;^B: PREVIOUS MENU
    bsr  display
    rts
;
;*************************
; subroutine menu1
; purpose: to display 1st menu
;*****************************
;
menu1:
    bsr  clear
    moveq     #line1,d3
    lea  messg36(pc),a5 ;0: reset
    bsr  display
    moveq     #line2,d3
    lea  messg37(pc),a5 ;1: display on/off
    bsr  display
    moveq     #line3,d3
    lea  messg61(pc),a5 ;^C : return to
    bsr  display
    moveq     #line4,d3
    lea  messg62(pc),a5 ;    menu
    bsr  display
    rts
;*********************
; subroutine menu2
; purpose: to display 2nd menu
;*****************************
;
menu2:
    bsr  clear
    moveq     #line1,d3
    lea  messg40(pc),a5 ;2:chan 1 high/low
    bsr  display
    moveq     #line2,d3
    lea  messg41(pc),a5 ;3:chan 3 high/low
    bsr  display
    moveq     #line3,d3
    lea  messg42(pc),a5 ;4 overload test on/off
```

```
    bsr  display
    moveq     #line4,d3
    lea  messg43(pc),a5 ;5 numerator (s/f)
    bsr  display
    rts
;*************************
; subroutine menu3
; purpose: to display 3rd menu
;*************************
menu3:
    bsr  clear
    moveq     #line1,d3
    lea  messg44(pc),a5 ;6: denominator (s/f)
    bsr  display
    moveq     #line2,d3
    lea  messg45(pc),a5 ;7: trig src chan1/2
    bsr  display
    moveq     #line3,d3
    lea  messg46(pc),a5 ;8: clear data store
    bsr  display
    moveq     #line4,d3
    lea  messg47(pc),a5 ;9: calibrate rms
    bsr  display
    rts
;
;*************************
; subroutine menu4
; purpose: to display 4th menu
;*******************************
;
menu4:
    bsr  clear
    moveq     #line1,d3
    lea  messg3(pc),a5  ;a: recal(dc only)  b: sample
    bsr  display
    moveq     #line2,d3
    lea  messg4(pc),a5  ;c: to calibrate
    bsr  display
    moveq     #line3,d3
    lea  messg5(pc),a5  ;d: sample size
    bsr  display
    moveq     #line4,d3
    lea  messg6(pc),a5  ;e: delay  f: freq
    bsr  display
    rts
;
;*************************
; subroutine menu5
; purpose: to display 5th menu
;*******************************
;
menu5:
```

```
    bsr  clear
    moveq     #line1,d3
    lea  messg48(pc),a5 ;^0:ENTER SHUNT VALS
    bsr  display
    moveq     #line2,d3
    lea  messg49(pc),a5 ;^1:LINES IN RAM
    bsr  display
    moveq     #line3,d3
    lea  messg50(pc),a5 ;^2:DISPLAY RAM DATA
    bsr  display
    moveq     #line4,d3
    lea  messg51(pc),a5 ;^3:STORAGE ON/OFF
    bsr  display
    rts
;
;*************************
; subroutine menu6
; purpose: to display 6th menu
;*****************************
;
menu6:
    bsr  clear
    moveq     #line1,d3
    lea  messg52(pc),a5 ;^4:KEYPAD PROGRAM
    bsr  display
    moveq     #line2,d3
    lea  messg53(pc),a5 ;^5:
    bsr  display
    moveq     #line3,d3
    lea  messg54(pc),a5 ;^6:
    bsr  display
    moveq     #line4,d3
    lea  messg55(pc),a5 ;^7:
    bsr  display
    rts
;
;*************************
; subroutine menu7
; purpose: to display 7th menu
;*****************************
;
menu7:
    bsr  clear
    moveq     #line1,d3
    lea  messg56(pc),a5 ;^8:
    bsr  display
    moveq     #line2,d3
    lea  messg57(pc),a5 ;^9:
    bsr  display
    moveq     #line3,d3
    lea  messg58(pc),a5 ;^A:
    bsr  display
```

```
    moveq     #line4,d3
    lea  messg59(pc),a5 ;^D:  ^E:
    bsr  display
    rts
;
;*************************************
; subroutine store_fact
; purpose     to update the numerators of the voltage,
;       current and power scale factors
; registers
; affected:   none
;*************************************
;
store_fact:
    move.w    tnumer_i1,numer_i1
    move.w    tnumer_i2,numer_i2
    btst.b    #6,state1
    bne.s     store_fact0
    move.w    tnumer_w1l,numer_w1 ;channel 1 on low range
    move.w    tnumer_v1l,numer_v1
    bra.s     store_fact1
store_fact0:
    move.w    tnumer_w1h,numer_w1 ;channel 1 on high range
    move.w    tnumer_v1h,numer_v1
store_fact1:
    btst.b    #7,state1
    bne.s     store_fact2
    move.w    tnumer_w2l,numer_w2 ;channel 3 on low range
    move.w    tnumer_v2l,numer_v2
    bra.s     store_fact3
store_fact2:
    move.w    tnumer_w2h,numer_w2 ;channel 3 on high range
    move.w    tnumer_v2h,numer_v2
store_fact3:
    rts
;*******************************
; subroutine init_fact
; purpose:    to initialize numerators
;       for power scale factor
; initial     a1 , a2 point to words to
; conditions: be multiplied
; final
; conditions: result stored in word at a3
; registers
; affected:   none
;*************************************
;
init_fact:
    movem.l   d0/d1,-(sp)
    move.w    (a1),d0
    move.l    #100,d1
    mulu.w    (a2),d0
```

```
    asr.l     #5,d0
    divu d1,d0
    tst.l     d0
    bpl.s     init_fact1
    addq.w    #1,d0
init_fact1:
    move.w    d0,(a3)
    movem.l   (sp)+,d0/d1
    rts
;
;************************************
; subroutine program
; purpose :to load the key buffer with the program buffer
;        contents and restore local mode when the buffer
;        code has been completely executed
; registers
; affected:   none
;************************************
;
program:
    movem.l   a0/a4,-(sp)
    bset.b    #6,pia_a  ;turn progrm mode execution led on
    move.b    pia_a,piada
    lea.l     key_buff,a0
    move.l    prog_pointer,a4      ;in program mode
    move.b    (a4)+,(a0)
    move.l    a4,prog_pointer
    cmpa.l    prog_max,a4
    bmi.s     program0
    clr.b   prog_state
    ori.b   #local,pia_b
    bclr.b    #7,pia_b  ;turn off remote led (no 8)
    move.b  pia_b,piadb
    bclr.b    #6,pia_a  ;turn progrm mode execution led off
    move.b    pia_a,piada
    move.l    #prog_buff,prog_max
    move.l    #prog_buff,prog_pointer
program0:
    movem.l   (sp)+,a0/a4
    rts
;
;*******************************************************
; subroutine automode
; purpose     to store device codes in prog_buffer
;        via keypad(keypad programming) and
;        thus simulate gpib programming
; registers
; affected:   none
;*******************************************************
;
automode:
    movem.l   d0/d1/a0-a5,-(sp)
```

```
    clr.l     d0
    clr.l     d1              ;character counter
    move.l    #prog_buff,prog_max ;initialize program mode
    move.l    #prog_buff,prog_pointer  ;variables
    lea  key_buff,a0
    move.b    #$ff,(a0)
        lea     disp,a1
        lea     disp+2,a2
  move.l   prog_max,a3
    bsr  clear
automode1:
        lea.l   prog_table-1(pc),a4
        lea.l   key_table(pc),a5
automode2:
    cmpi.b    #$9b,(a0)
    bhi.s     automode2 ;out of range
    move.b    (a0),d0
    cmpi.b    #$9a,d0         ;'U to undo
    bne.s     automode4
    tst.b     d1
    beq.s     automode2 ;no character entered
    subq.l    #1,a3
    move.l    a3,prog_max
    move.b    #$11,(a1)
automode3:
    tst.b     (a1)
    bmi.s     automode3
    move.b    #$ff,(a0)
    bra.s     automode1
automode4:
    cmpi.b    #$99,d0         ;'T to end (do not store)
    beq.s     automode7
automode5:
        addq.l  #1,a4
        cmp.b   (a5)+,d0
        bne.s   automode5
        move.b  d0,(a3)+
    move.l    a3,prog_max
    addi.b    #1,d1
    move.b    (a4),(a2)
automode6:
    tst.b     (a1)
    bmi.s     automode6 ;wait till display ready
    move.b    #$ff,(a0) ;prepare to receive new byte
    move.b  #$ff,prog_state
    bra.s     automode1
automode7:
    move.b    #$ff,(a0)
    movem.l   (sp)+,d0/d1/a0-a5
    rts
;
;
```

```
; subroutine lines_left
; purpose:    to display the number of lines remaining
;       in ram for ~ 2.5 secs
; registers
; affected: none
;
lines_left:
    movem.l   d0-d3/a5/a6,-(sp)
    bsr  clear
    clr.l     d0
    clr.l     d1
    move.w    store_size,d1
    bsr  bcd1 ;convert to bcd
    lea.l     disp_buff,a5
    move.b    #' ',d3
    move.w    #'LR',d2
    move.w    d2,(a5)+
    bsr  update
    bsr  blank
    moveq     #line3,d3
    lea.l     disp_buff,a5
    bsr  display
    moveq     #line1,d3
    lea  messg46a,a5
    bsr  display
    move.l    #1111111,d0
lines_left1:
    subq.l    #1,d0
    bne.s     lines_left1    ;wait for ~ 2.5 secs
    movem.l   (sp)+,d0-d3/a5/a6
    rts
;
; subroutine disp_data
; purpose: to display   a)No of lines of data remaining at
;             at ~ 2.5 secs intervals.
;             b)Data at ~ 5 secs intervals
;
disp_data:
    movem.l   d0-d3/a0-a3/a5,-(sp)
    bclr.b    #7,pia_a  ;turn ram full led off
    move.b    pia_a,piada
    lea.l     key_buff,a0
    lea.l     data_pointer+4,a1
    lea.l     store_size,a2
disp_data0:
    bsr  lines_left
    bsr  clear
    tst.w     (a2)
    beq  disp_data6b    ;no stored data
    clr.l     d0
    move.b    (a1)+,d0  ;obtain measurement state
    clr.b     meas_state1
```

```
    jmp  disp_data1(pc,d0.w)
;
disp_data1:
    bra.s     disp_data1a     ;ac_calibrate
    bra.s     disp_data1b     ;dc_calibrate
    bra.s     disp_data1c     ;calibrate rms
    bra.s     disp_data1d     ;normal ac
    bra.s     disp_data1e     ;normal dc
    bra.s     disp_data1f     ;3-phase ac
    bra.s     disp_data1e     ;3-phase dc
    bra.s     disp_data1g     ;frequency
;
disp_data1a:
    lea.l     messg9(pc),a5   ;calibrate ac
    bra.s     disp_data1i
disp_data1b:
    lea.l     messg10(pc),a5 ;calibrate dc
    bra.s     disp_data1i
disp_data1c:
    lea.l     messg9a(pc),a5 ;calibrate rms
    bra.s     disp_data1i
disp_data1d:
    bra.s     disp_data2
disp_data1e:
    lea.l     messg10a(pc),a5      ;dc
    bra.s     disp_data1k
disp_data1f:
    bra.s     disp_data2
disp_data1g:
    lea.l     messg8(pc),a5   ;frequency in hertz
    bra.s     disp_data1k
disp_data1i:
    moveq     #line1,d3
    bsr  display
    lea.l     disp_buff2,a3
disp_data1j:
    move.b    (a1)+,(a3)+     ;fill buffer till termination
    bne.s     disp_data1j     ;charater(0) is encountered
    subq.w    #1,(a2)         ;update store_size
    moveq     #line3,d3
    lea.l     disp_buff2,a5
    bsr  display
    bra.s     disp_data6
disp_data1k:
    moveq     #line1,d3
    bsr  display
    move.b    #1,meas_state1
    bra.s     disp_data3
disp_data2:
    lea.l     disp_buff,a3
disp_data2a:
    move.b    (a1)+,(a3)+     ;fill buffer till termination
```

```
        bne.s     disp_data2a     ;charater(0) is encountered
        subq.w    #1,(a2)         ;update store_size
    ;
    disp_data3:
        lea.l     disp_buff1,a3
    disp_data3a:
        move.b    (a1)+,(a3)+     ;fill buffer till termination
        bne.s     disp_data3a     ;charater(0) is encountered
        subq.w    #1,(a2)         ;update store_size
    disp_data4:
        lea.l     disp_buff2,a3
    disp_data4a:
        move.b    (a1)+,(a3)+     ;fill buffer till termination
        bne.s     disp_data4a     ;charater(0) is encountered
        subq.w    #1,(a2)         ;update store_size
    disp_data5:
        lea.l     disp_buff3,a3
    disp_data5a:
        move.b    (a1)+,(a3)+     ;fill buffer till termination
        bne.s     disp_data5a     ;charater(0) is encountered
        subq.w    #1,(a2)         ;update store_size
        cmpi.b    #1,meas_state1
        beq.s     disp_data5b
        moveq     #line1,d3
        lea.l     disp_buff,a5
        bsr  display
    disp_data5b:
        moveq     #line2,d3
        lea.l     disp_buff1,a5
        bsr  display
        moveq     #line3,d3
        lea.l     disp_buff2,a5
        bsr  display
        moveq     #line4,d3
        lea.l     disp_buff3,a5
        bsr  display
    disp_data6:
        move.l    #2222222,d0
    disp_data6a:
        subq.l    #1,d0
        bne.s     disp_data6a     ;wait for ~ 5 secs
        cmpi.b    #$83,(a0)
        bne.s     disp_data7
        bra  disp_data0
    disp_data6b:
        move.b    #$ff,(a0) ;normal exit
    disp_data7:
        clr.w     (a2)       ;clear data store
        move.l    #data_pointer+4,data_pointer  ;initialize
    data pointer
        bclr.b    #0,state
        movem.l   (sp)+,d0-d3/a0-a3/a5
```

```
    rts
;
;*****************************
; subroutine store_data
; purpose: to store displayed data in ram
; registers
; affected: none
;*****************************
;
store_data:
    movem.l   d0/a0-a2,-(sp)
    clr.l     d0
    lea.l     data_pointer,a2
    lea.l     store_size,a3
    move.l    (a2),a0
    move.b    meas_state,d0
    move.b    d0,(a1)+  ;store measurement state
    jmp  store_data0(pc,d0.w)
store_data0:
    bra.s     store_data1     ;ac_calibrate
    bra.s     store_data1     ;dc_calibrate
    bra.s     store_data1     ;calibrate rms
    bra.s     store_data1     ;normal ac
    bra.s     store_data2     ;normal dc
    bra.s     store_data1     ;3-phase ac
    bra.s     store_data2     ;3-phase dc
    bra.s     store_data2     ;frequency
store_data1:
    lea.l     disp_buff,a0
store_data1a:
    move.b    (a0)+,(a1)+
    bne.s     store_data1a
    addq.w    #1,(a3)
    cmpi.w    #4,d0
    bls.s     store_data5
store_data2:
    lea.l     disp_buff1,a0
store_data2a:
    move.b    (a0)+,(a1)+
    bne.s     store_data2a
    addq.w    #1,(a3)
store_data3:
    lea.l     disp_buff2,a0
store_data3a:
    move.b    (a0)+,(a1)+
    bne.s     store_data3a
    addq.w    #1,(a3)
store_data4:
    lea.l     disp_buff3,a0
store_data4a:
    move.b    (a0)+,(a1)+
    bne.s     store_data4a
```

```
    addq.w    #1,(a3)
store_data5:
    move.l    a1,(a2)
    cmpa.l    #stack_base,a1
    bmi.s     store_data5a
    bclr.b    #4,state1
    bclr.b    #5,pia_a
    bset.b    #7,pia_a
    move.b    pia_a,piada
store_data5a:
    movem.l   (sp)+,d0/a0-a2
;
    rts
;
;**************************************
; subroutine ov_load
; purpose: to indicate overload
;      on any channel
;**************************************
;
ov_load:
    movem.l   d0-d4/a1/a2/a5,-(sp)
    lea  $200008,a1      ;point to Y0 chan 1
    lea  state1,a2
    bclr.b    #1,(a2)         ;clear overload bit
    clr.l     d0
    clr.l     d1
    clr.l     d4
    move.l    #$7fe,d2
    move.l    #$ffe,d3
    btst.b    #7,state
    bne.s     ov_load1a
    btst.b    #0,state
    beq.s     ov_load1a
    sne.b     d4
ov_load1:
    move.w    (a1)+,d0
    beq.s     ov_load3  ;overload
    cmp.w     d0,d3
    bls.s     ov_load3  ; overload
    bra.s     ov_load2a ; no overload
ov_load1a:
    move.w    (a1)+,d0
    bpl.s     ov_load2
    neg.w     d0        ;return absolute value
ov_load2:
    cmp.w     d0,d2
    bls.s     ov_load3  ; overload
ov_load2a:
    addq.b    #1,d1     ;point to next channel
    bclr.l    #2,d1
    tst.b     d4
```

```
        bne.s     ov_load2b
        cmpa.l    a5,a1
        bls.s     ov_load1a
        bra.s     ov_load4
ov_load2b:
        cmpa.l    a5,a1
        bls.s     ov_load1
        bra.s     ov_load4
ov_load3:
        bsr  clear
        bset.b    #1,(a2)
        add.w     #' 1',d1  ;determine correct channel
        moveq.l   #line1,d3
        lea  messg35(pc),a5
        bsr  display
        lea  disp_buff,a5
        move.l    #' CHA',(a5)+
        move.l    #'NNEL',(a5)+
        move.w    d1,(a5)+
        move.b    #0,(a5)
        moveq     #line3,d3
        lea  disp_buff,a5
        bsr  display
ov_load4:
        movem.l   (sp)+,d0-d4/a1/a2/a5
        rts
;
;*********************
; subroutine  ac_calibrate
; registers
; affected : d0-d2,a0,a4,a5
; result in d0 and appropriate cal_buff
;*******************
;
ac_calibrate:
        move.b  cal_chan,d0
        asl.l   #1,d0
        addq.l  #6,a1
        adda.l  d0,a1   ;point to Y1
        lea     cal_buff1-2,a0
        adda.l  d0,a0
        suba.l  #16,a5
        adda.l  d0,a5   ;point to Yn-1
        clr.l   d0
        clr.l   d2
        move.w  8(a5),d2        ;Yn
        add.w   -8(a1),d2       ;Y0
        move.w  #16,d3
        add.w   alpha16_store,d3 ; 16 + 16*delta
        muls    d3,d2           ;(16+16*delta)*(Y0+Yn)
        asr.l   #1,d2           ;0.5*(16+16*delta)*(Y0+Yn)
calib_ac1:
```

```
        move.w  (a5),d1
        add.l   d1,d0
        subq.l  #8,a5
        cmpa.l  a5,a1
        bls.s   calib_ac1
        asl.l   #4,d0           ;16*d0
        add.l   d2,d0
        moveq.l #31,d1
        add.w   alpha16_store,d5  ;16*n + 16*delta
        divu    d5,d0
        btst.l  d1,d0
        beq.s   calib_ac2
        addq.w  #1,d0    ;round up
calib_ac2:
        move.w  d0,(a0)
        swap    d0
        clr.w   d0
        swap    d0
;
        rts
;
;*********************
; subroutine  ac_rms
; purpose: to compute and display rms value
;      in the calibration mode
; registers
; affected : d0-d2,a0,a4
; result in d0
;*******************
;
ac_rms:
        move.b  cal_chan,d0
        asl.l   #1,d0
        addq.l  #6,a1
        adda.l  d0,a1   ;point to Y1
    movea.l   a1,a4     ;initialize a4 (Y1)
    movea.l   a5,a2
    subq.l    #8,a2
    adda.l    d0,a2     ;initialize a2 (Yn)
    movea.l   a2,a3     ;and a3 (Yn)
    bsr  correct
    bsr  evaluate
    move.l    4(a0),d0
    bsr  sqrt
    rts
;
;*************************
; subroutine  dc_calib
; registers
; affected: d0,d2,a0,a1,a2,a5
; result in d0 and appropriate
;       cal_buff
```

```
;***************************
;
dc_calib:
        move.b  cal_chan,d0
        asl.l   #1,d0
        subq.l  #2,a1
        adda.l  d0,a1    ;point to data start (Y0)
        lea     cal_buff1-2,a0
        adda.l  d0,a0
        subq.l  #8,a5
        adda.l  d0,a5
        bsr     dc_avrg
    btst.b    #7,state
    beq.s     dc_calib2
;
    addq.w    #2,recal_point
    move.w    recal_point,d1
    lea  recal_buff-2,a2
    move.w    d0,(a2,d1.w)
    cmpi.w    #512,recal_point     ;256 samples processed?
    bcs.s     dc_calib3
;
;  compute average
;
    lea  recal_buff,a1
    lea  recal_buff+510,a5
;
    clr.l     d0
    clr.l     d1
dc_calib1:
    move.w    (a1)+,d1
    ext.l     d1
    add.l     d1,d0
    cmpa.l    a5,a1
    bls.s     dc_calib1
;
    asr.l     #8,d0                ;divide by 2^8
    add.w     (a0),d0
    btst.b    #0,state1       ;in autocal mode?
    beq.s     dc_calib1a
    move.l    auto_chan,d1
    lsl.l     #8,d1
    move.l    d1,auto_chan
    tst.l     auto_chan
    beq.s     dc_calib1a
    move.b    #$9c,key_buff
    bra.s     dc_calib2
dc_calib1a:
    move.b    #false_key,key_buff
    bclr.b    #0,state1
dc_calib2:
    move.w    d0,(a0)
```

```
dc_calib3:
        rts
;
;*****************************
; SUBROUTINE: dc_avrg
;
; purpose:    to compute the average of N values
;        using the trapezoidal rule.
; initial     a1 points to Y0
; conditions:      a5 points to Yn
; final       result in d0
; conditions:
; registers
; affected: d0
;*****************************
;
dc_avrg:
        movem.l d1/a1/a5,-(sp)
        clr.l   d0
        move.w  (a1),d0      ;Y0
    ext.l     d0
    move.w    (a5),d1
    ext.l     d1
        add.l   d1,d0        ;Y0 + Yn
        asr.l   #1,d0           ;0.5(Y0 + Yn)
    clr.l     d1
        adda.l  #8,a1           ;point to Y1
dc_avrg1:
        suba.l  #8,a5
        move.w  (a5),d1
        ext.l   d1
        add.l   d1,d0
        cmpa.l  a5,a1
        bcs.s   dc_avrg1
        moveq.l #10,d1       ;1024 values(2^10)
        asr.l   d1,d0
        bcc.s   dc_avrg2
        addq.l  #1,d0      ;round up
;
dc_avrg2:
        movem.l (sp)+,d1/a1/a5
        rts
;
;***********************
; subroutine init_samp
;***********************
;
init_samp:
    move.w    #$e,d0    ;tim_latch mask value
        moveq   #0,d1
    moveq     #0,d7
        lea     tim_latch,a0
```

```
        lea     adc1,a1
        lea     adc2,a2
        lea     adc3,a3
        lea     adc4,a4
        lea     samp_st,a5
        rts
;
;*****************************
; subroutine init_sampac
;*****************************
;
init_sampac:
        andi.b  #stac_dis,pia_b
        move.b  pia_b,piadb     ;reset flip-flops also
;                               start' and ac' commands
        move.b  #%10100011,time+2
        move.b  #%11000010,time   ;timer 1 continous mode
;                                     start all timers
        ori.b   #stac_en,pia_b  ;start and ac from pia
        andi.b  #key_dis,pia_b
        move.b  pia_b,piadb
        lea.l   ac_mode,a6
        move.b  #$ff,(a6)       ;assume ac mode
    move.l    #smul_store,d6
    subq.l    #2,d6           ;point to smul_store-2
    rts
;
;******************
; subroutine clear
; purpose: to clear the display
;******************
;
clear:
    movem.l   d0/a0,-(a7)
    lea       disp,a0
    move.b    #1,(a0)
clear1  move.b    (a0),d0
        bmi.s     clear1
        movem.l   (a7)+,d0/a0
        rts
;
;******************
; subroutine clear_regs
; purpose: to clear registers d0-d3
; registers
; affected:   d0-d3
;******************
;
clear_regs:
    moveq.l   #0,d0
    moveq.l   #0,d1
    moveq.l   #0,d2
```

```
    moveq.l    #0,d3
;
    rts
;
;*********************************
; subroutine factor
; purpose: to increment/decrement scale factors
;      for voltage and currents for all 4 channels
; initial     a1 points to base location -4
; conditions: a0 points to the key buffer
;        d2.w holds ASCII N0/D0 (numerator/denominator)
; registers   d0,d1,a2
; affected:   a6    update
;*********************************
;
factor:
    bsr  cal_display
    clr.l      d0
factora:
    move.b     (a0),d0
    beq.s      factora
factorb:
    cmpi.b  #$a,d0
        bhi.s    factora
        lea      key_table+7(pc),a2
        moveq    #6,d1
factorc:
    cmp.b    -(a2),d0
        dbeq     d1,factorc
    bsr  clear
    move.b     #$ff,(a0)
    add.b      d1,d2
    lsl.l      #2,d1
    adda.l     d1,a1
    bra.s      factor4
factor1:
    cmpi.b     #$1b,(a0) ;f pressed?
    bne.s      factor2
    addq.w     #1,(a1)
    bra.s      factor4
factor2:
    cmpi.b     #$13,(a0) ;b pressed?
    bne.s      factor3
    subq.w     #1,(a1)
    bra.s      factor4
factor3:
    cmpi.b     #$8a,(a0)
    bne.s      factor1
    move.b     #$ff,(a0)
    bra.s      factor5
factor4:
    move.b     #$ff,(a0)
```

```
    clr.l     d0
    clr.l     d1
    move.w    (a1),d1
    bsr  bcd1       ;convert to bcd
    lea.l     disp_buff,a5
    move.b    #' ',d3
    move.w    d2,(a5)+
    bsr  update
    bsr  blank
    moveq.l   #line2,d3
    lea.l     disp_buff,a5
    bsr  display
    bra.s     factor1
factor5:
    bclr.b    #0,state
    rts
;
;**********************
; subroutine cal_display
; purpose: to display messages
;          in calibration mode
; registers
; affected: none
;**********************
;
cal_display:
        bsr       clear
    bset.b    #0,state
        moveq     #line1,d3
        lea       messg13(pc),a5
        bsr       display
        moveq     #line2,d3
        lea       messg14(pc),a5
        bsr       display
        moveq     #line3,d3
        lea       messg15(pc),a5
        bsr       display
        moveq     #line4,d3
        lea       messg16(pc),a5
        bsr       display
    move.b    #$ff,(a0)
        rts
;
;**********************************
; SUBROUTINE acq_disp
; purpose: to display acquisition menu
;**********************************
;
acq_disp:
    move.l    d3,-(sp)
        moveq     #line1,d3
        lea       messg22(pc),a5
```

```
        bsr     display
    btst.b  #4,state1
    beq.s   acq_disp0
    btst.b  #5,state1
    bne.s   acq_disp1
        moveq   #line2,d3
        lea     messg26(pc),a5
        bsr.s display
    bra.s   acq_disp2
acq_disp0:
    btst.b  #5,state1
    bne.s   acq_disp1
        moveq   #line2,d3
        lea     messg23(pc),a5
        bsr     display
    bra.s   acq_disp2
acq_disp1:
        moveq   #line2,d3
        lea     messg23a(pc),a5
        bsr     display
acq_disp2:
    move.l  (sp)+,d3
        rts
;
;*****************************
;  subroutine display
;  purpose: to display an ascii string
;  initial     d3 points to current line to be displayed
;  conditions: a5 points to start of string
;  registers used: d0,d3,a0,a1
;  registers affected: none
;*****************************
;
display:
    movem.l d0/d3/a0/a1/a5,-(sp)
        lea     disp,a0
        lea     disp+2,a1
        addi.b  #$80,d3
        move.b  d3,(a0)    ;set dd ram address
        bra.s   busy_d
disp1   move.b  (a5)+,(a1)
busy_d  tst.b   (a0)     ;also performs a read
        bmi.s   busy_d
        tst.b   (a5)    ;end of string?
        bne.s   disp1
    movem.l (sp)+,d0/d3/a0/a1/a5
        rts
;
;************************
; SUBROUTINE: sign
; purpose: to determine sign
;          of result
```

```
;
sign:
    move.b  #' ',d3
        tst.l   d0
        bpl.s   sign0
        neg.l   d0              ;negative value
        move.b  #'-',d3
sign0   rts
;
;**************************
; subroutine  correct
; purpose : to calculate 0.5(16+16*delta)(Y0+Yn)
; registers
; affected: d0-d3
; result in : d2 -d3
;**************************
;
correct:
    movem.l  d4/d5,-(sp)
        clr.b   d4
        move.w  (a2),d0
        muls    (a3),d0         ;Yn
        move.w  -8(a1),d1
        muls    -8(a4),d1       ;Y0
        add.l   d1,d0
        bpl.s   correct0
        smi.b   d4
        neg.l   d0
correct0:
    clr.l     d1
        move.w  alpha16_store,d1
        add.w   #16,d1
    bpl.s     correct1
    smi.b     d5
    neg.w     d1
    ext.l     d1
correct1:
        bsr.s mul32_32
        asr.l   #1,d0   ; divide by 2
        roxr.l  #1,d1
    cmp.b     d4,d5
        beq.s   correct2        ;positive result?
        neg.l   d1
        negx.l  d0
correct2:
        movem.l  (sp)+,d4/d5
        move.l  d0,d2
        move.l  d1,d3
        rts
;
;
;
```

```
; subroutine mul32_32
; purpose :to perform an unsigned multiplication of
;          a 32-bit value by a 32-bit value
; initial     32_bit number in d0 (A B)
; conditions: 32_bit number in d1 (C D)
; final conditions: result in d0-d1
; registers
; affected:   none
;
mul32_32:
    movem.l   d2-d4,-(sp)
    move.l    d1,d2           ;C D
    move.l    d1,d3           ;C D
    swap d3         ;D C
    move.l    d0,d4           ;A B
    swap d4         ;B A
    mulu d0,d1           ;pp1=B*D
    mulu d3,d0           ;pp2=B*C
    mulu d4,d2           ;pp3=A*D
    mulu d4,d3           ;pp4=A*C
    swap d1         ;sum1=pp2 low +
    add.w     d0,d1           ;pp1 high
    clr.l     d4
    addx.l    d4,d3           ;propagate carry into pp4
    add.w     d2,d1           ;sum2=sum1 + pp3 low
    addx.l    d4,d3           ;propagate carry into pp4
    swap d1         ;put low prod. in correct order
    clr.w     d0         ;prepare for sum3
    swap d0
    clr.w     d2
    swap d2
    add.l     d2,d0           ;sum3=pp2 high + pp3 high
    add.l     d3,d0           ;sum4=sum3 + pp4
    movem.l   (sp)+,d2-d4
        rts
;
;*****************************
; subroutine smuladd
; purpose :to return the sum of the product of 2
;          16_bit values added to a 64-bit value
;          in d0-d1
; initial    a2 points to end of 2nd(low address) string+8
; conditions:a3 points to end of 2nd(high address) string+8
;            a1 points to start of 1st(low address) string
;            d0-d1 contains initial 64-bit value
; final
; condition: 64-bit signed result in d0-d1
; registers
; affected: d0,d1
;*****************************
;
smuladd:
```

```
    movem.l d2-d3/a2-a3,-(sp)
smuladd0:
        subq.l  #8,a2
        subq.l  #8,a3
        moveq.l #-1,d2
        move.w  (a2),d3
        muls.w  (a3),d3
        bmi.s   smuladd1
        clr.l   d2
smuladd1:
        add.l   d3,d1
        addx.l  d2,d0
        cmpa.l  a1,a2
        bhi.s   smuladd0
        movem.l (sp)+,d2-d3/a2-a3
        rts
;
;
;*********************************
; SUBROUTINE  update
; purpose : to update display
;           for computed parameters
; initial     d0 contains number in hex
; conditions: d3 contains sign
; registers
; affected: a6
;*********************************
;
update:
    movem.l  d5/a5,-(sp)
    btst.b    #3,state
    bne.s     update_4
    btst.b    #0,state
    bne.s     update_3
update_0:
    btst.b    #1,state
    beq.s     update_1
    move.b    #'1',(a5)+      ;Phase 1
    bra.s     update_3
update_1:
    btst.b    #2,state
    bne.s     update_2
    movea.l   #5,a2
    bra  ramerr
update_2:
    move.b    #'2',(a5)+
update_3:
        move.w  #'= ',(a5)+
update_4:
        move.b  #' ',(a5)+
        move.b  d3,(a5)+
        move.l  d0,(a5)
```

```
        moveq   #4,d5
        bsr     hex_ascii
        movem.l  (sp)+,d5/a5
        rts
;
;***************************
; SUBROUTINE BLANK
; purpose :   to blank off up to 6 leading zeroes
;       from a result in a display buffer.
; initial
; conditions:      a6 points to 0 termination char location
; registers
; affected:   none
;***************************
;
blank:
    movem.l   d2/a5/a6,-(sp)
    movea.l   a6,a5
    subq.l    #8,a5
    moveq.l   #5,d2
blank_1:
    cmpi.b    #'0',(a5)+
    bne.s     blank_2
    move.b    #' ',-1(a5)     ;replace with blank
    dbra d2,blank_1
blank_2:
    movem.l   (sp)+,d2/a5/a6
    rts
;
;***********************************
; SUBROUTINE  EVALUATE
; purpose to evaluate mean square/power
; registers affected:d0-d3,a0
; final        result in 64-bit location smul_store
; conditions: (pointed to by a0)
;***********************************
;
evaluate:
        move.l  d5,-(sp)
        clr.l   d0
        clr.l   d1
        bsr     smuladd   ;compute  sum of products
;                         for reqd. pair of channels
        move.l  d2,-(sp)
        moveq.l #3,d2    ;4 shifts
;
eval_0  lsl.l   #1,d1
        dbra    d2,eval_0       ;*16
        move.l  (sp)+,d2
        add.l   d3,d1
        addx.l  d2,d0
        move.b  #' ',d3
```

```
        tst.l   d0
        bpl.s   eval_1
        move.b  #'-',d3
        neg.l   d1          ;compute 2's complement
        negx.l  d0
eval_1  lea     smul_store,a0
        add.w   alpha16_store,d5        ; 16n + 16*delta
        move.l  d5,d0
        moveq.l #4,d2
        bsr.s divide    ;divide above by
;                           number of samples
        move.l  (sp)+,d5
        rts
;
;************************************************
; subroutine divide
; unique features:a)no overflow occurs
; purpose:to divide a n x16-bit unsigned integer
;        by a 16-bit unsigned integer to produce
;   an nx16-bit quotient and a 16-bit remainder.
; initial     a)a0 points to the dividend's most
; conditions:  significant word
;             b)the divisor is in d0
;             c)the dividend lenght in words
;               is in d2
; final       a)a0 points to the quotient's most
; conditions:   significant word
;             b)remainder in lower word of d0
; registers
; affected:   d0,d2
;************************************************
;
;
divide:
    movem.l d1/a0,-(sp)
        clr.l   d1          ;prepare work register
        subq.l  #1,d2        ;initiate words
;                            counter in d2
divide1
    move.w  (a0)+,d1    ;next word of dividend
;                           dividend in d1
        divu    d0,d1      ;division of word by d0
        move.w  d1,-(a0)   ;save partial quotient
        dbra    d2,divide1
        move.w  d1,d0       ;store remainder
        movem.l (sp)+,d1/a0
        rts
;
;****************************************
;  subroutine sqrt
;  purpose: to calculate the square root
;           of a 32-bit number in d0
```

```
;           (max: $fffffffe )
;  initial
;  conditions: 32-bit number in d0
;  final
;  conditions: square-root in d0
;  actual
;  square-root: [result-1,result]
;  register
;  affected:    d0
;
;  sample results
;     initial            final
;      value             value
;        1                 1
;        4                 2
;        7                 3
;  $fffffffe           $10000
;  $ffffffff           $10000
;
;****************************************
;
sqrt:
    cmpi.l  #1,d0
        bls     sqrt3   ;square-root of 0/1=0/1
    cmpi.l    #-1,d0
    bne.s     sqrt0
    move.l    #$10000,d0
    bra  sqrt3
sqrt0    movem.l   d1/a0/a1,-(a7)
        lea     sqr(pc),a0
        lea     root(pc),a1
        moveq   #-4,d2
sqrt1   addq.l  #4,d2
        cmp.l   (a0)+,d0
        bcc.s   sqrt1
        move.l  (a1,d2.w),d1    ;determine offsets
        sub.l   -4(a0),d0       ;required for
        add.l   (a1,d2.w),d0    ;registers d0,d1
        addq.l  #2,d0
sqrt2   addq.l  #2,d1
        sub.l   d1,d0
        bhi.s   sqrt2
        addq.l  #1,d1
        lsr.l   #1,d1           ;obtain square-root
        move.l  d1,d0
        movem.l (a7)+,d1/a0/a1
        bra     sqrt3
        dc.l    1                       ;special case
sqr     dc.l    $1000000,$4000000       ;square
        dc.l    $9000000,$10000000      ;offsets
        dc.l    $19000000,$24000000
        dc.l    $31000000,$40000000
```

```
        dc.l    $51000000,$64000000
        dc.l    $79000000,$90000000
        dc.l    $a9000000,$c4000000
        dc.l    $e1000000,$ffffffff
root    dc.l    -1,$1fff,$3fff,$5fff
        dc.l    $7fff,$9fff,$bfff   ;square-root
        dc.l    $dfff,$ffff,$11fff    ;offsets
        dc.l    $13fff,$15fff,$17fff
        dc.l    $19fff,$1bfff,$1dfff
;
sqrt3   rts
;
;
;*************************
; subroutine hex_ascii
; purpose:    to convert a hex string to ascii
; initial     d5 contains number of bytes excluding
; conditions: zero termination
;                 a5 points to start of string
; final  a5 points to start of ascii string
; conditions: a6  points  to  location  of  0  termination
character
; register
; affected:       a6
;*************************
;
hex_ascii:
        movem.l d0-d5/a0/a5,-(sp)
        movea.l a5,a0
        move.l  d5,d3
        adda.l  d5,a5     ;point to end of string+1
        lsl.l   #1,d3     ;convert to nibbles
        adda.l  d3,a0     ;point to end of result+1
        move.b  #0,(a0)
hex1    move.b  -(a5),d2
        andi.b  #$f,d0
        bsr.s   hex2
        lsr.b   #4,d2
        andi.b  #$f,d0
        bsr.s   hex2
        dbra    d5,hex1
        movem.l (sp)+,d0-d5/a0/a5
;
        rts
;
hex2    cmpi.b  #9,d0
        bls.s   hex3
        add.b   #'A'-'9'-1,d0
hex3    add.b   #'0',d0
        move.b  d0,-(a0)
        rts
;
```

```
;***********************
; subroutine bcd1
; purpose:to convert long word in d1
;         to bcd.
; initial
; conditions: long word in d1
; final
; conditions: result in d0
; registers
; affected: d0,d1
; max. bcd number : 999,999
;*****************************
;
;
bcd1:
    cmpi.l  #100000,d1
        bcs.s   bcd2
        subi.l  #100000,d1
        addi.l  #$100000,d0
        bra.s   bcd1
bcd2    cmpi.w  #10000,d1
        bcs.s   bcd3
        subi.w  #10000,d1
        addi.l  #$10000,d0
        bra.s   bcd2
bcd3    cmpi.w  #1000,d1
        bcs.s   bcd4
        subi.w  #1000,d1
        addi.w  #$1000,d0
        bra.s   bcd3
bcd4    cmpi.w  #100,d1
        bcs.s   bcd5
        subi.w  #100,d1
        addi.w  #$100,d0
        bra.s   bcd4
bcd5    cmpi.w  #10,d1
        bcs.s   bcd6
        subi.w  #10,d1
        addi.w  #$10,d0
        bra.s   bcd5
bcd6    add.w   d1,d0
;
        rts
;
;*************************
; SUBROUTINE: ACQ
; initial
; conditions: a1 points to final store
;         a0 points to key_buff
; registers
; affected:
;*************************
```

```
acq:
    movem.l   d0-d2/a2-a4,-(sp)
        lea     disp,a2
        lea     disp+2,a3
        move.b  #$da,(a2)
acq_0   tst.b   (a2)
        bmi.s   acq_0
        moveq   #3,d2
acq_1   tst.b   prog_state
        beq.s   acq_1a
   bsr  program
acq_1a  move.b  (a0),d0
    cmpi.b    #$89,d0          ;^5 to CORRECT last ENTRY
    bne.s     acq_1c
    cmpi.b    #3,d2            ;no numeral entered ?
    beq.s     acq_1
    addq.w    #2,d2
    subq.l    #2,a1
    move.b    #$11,(a2)
acq_1b  tst.b     (a2)
    bmi.s     acq_1b           ;wait till display ready
    move.b  #$ff,(a0)          ;prepare to accept new byte
    bra.s     acq_1
acq_1c  cmpi.b  #$11,d0          ;neglect if > 9
        bhi.s   acq_1
        lea     key_table+10(pc),a5
        moveq   #7,d1
acq_2   cmp.b   -(a5),d0
        dbeq    d1,acq_2
        move.b  d1,(a1)+
        addi.b  #$30,d1          ;convert to ascii
        move.b  d1,(a3)          ;echo character
acq_3   tst.b   (a2)
        bmi.s   acq_3            ;wait till data accepted
        move.b  #$ff,(a0)
        dbra    d2,acq_1
acq_3a   tst.b   prog_state
        beq.s   acq_3b
    bsr  program
acq_3b   cmpi.b    #$89,(a0) ;^5 to CORRECT last ENTRY
    beq.s     acq_1a
    cmpi.b    #$8a,(a0) ;^6 to END ENTRY
    bne.s     acq_3a
        move.b  #$ff,(a0)
    subq.l    #4,a1
    move.l    (a1),d0
    cmpa.l    #acq_count,a1
    beq.s     acq_5            ;in acquisition mode
    cmpa.l    #delay_value,a1
    bne.s     acq_7            ;not in delay mode
    tst.l     d0         ; in delay mode
    bne.s     acq_4
```

```
        bclr.b    #5,state
        bra.s     acq_8
acq_4   bset.b    #5,state
        bra.s     acq_7
acq_5   tst.l     d0
        bne.s     acq_6
        bclr.b    #4,state
        bra.s     acq_8
acq_6   bset.b    #4,state
acq_7   bsr.s     dtohex
        move.w    d0,(a1)
acq_8   movem.l   (sp)+,d0-d2/a2-a4
        rts
;
;***************************
; SUBROUTINE : dtohex
; purpose: to convert 4 decimal digits
;          in d0 (one digit per byte)to hex
; initial
; conditions: decimal digits in d0 (0a0b0c0d)
; final
; conditions: hex word in d0
; registers
; affected: d0
;
; sample results:
;     initial              final word
;    09090909                 270f
;
;*****************************
;
dtohex:
        movem.l d1-d3,-(sp)
        move.l  d0,d1
        move.l  d0,d2
        swap    d2
        move.w  d2,d3
        andi.w  #$f,d0
        lsr.w   #8,d1
        andi.w  #$f,d2
        lsr.w   #8,d3
    tst.w     d1
    beq.s     dtohex1          ;trivial case
        mulu    #10,d1             ;d1=10*c
dtohex1:
    tst.w     d2
    beq.s     dtohex2          ;trivial case
        mulu    #100,d2
dtohex2:
    tst.w     d3
    beq.s     dtohex3          ;trivial case
        mulu    #1000,d3
```

```
dtohex3:
        add.w   d3,d0
        add.w   d1,d0           ;final result
        movem.l (sp)+,d1-d3
        rts
;
;**************************
;
; Subroutine : max_sample
; purpose: to return the sample number
;      corresponding to the maximum.
; initial     a0 points to location for result
; conditions: a1 points to Y1
;        d5 contains no of samples * 16
; final
; conditions: a0 points to location holding
;        sample number
; registers
; affected:   none
;
;****************************************
;
max_sample:
    movem.l   d0-d2/d5/a1,-(sp)
    clr.w     d1
    move.w    d1,d2
    lsr.w     #4,d5
    subq.w    #1,d5
    move.w    -8(a1),d0 ;Assume Y0 is max. value
    addq.w    #1,d1
    move.w    (a1),d0         ;update d0
max_0   addq.l    #8,a1
    addq.w    #1,d1
    cmp.w     (a1),d0
    bcc.s     max_1
    move.w    (a1),d0         ;update d0
max_1   dbra d5,max_0
    move.w    d2,(a0)         ;store max. sample no.
    movem.l   (sp)+,d0-d2/d5/a1
;
    rts
;
;**************************
; Subroutine : dc_power
; purpose: to compute the power the
;      dc case
; initial     a1 points to Y1 chan 1/3
; conditions: a2 points to Yn chan 1/3
;        a5 points to Yn chan 2/4
; final       result in d0
; conditions: ascii value of sign in d3
;
```

```
; registers
; affected:    d0,d3
;
;*****************************
;
dc_power:
    movem.l   d1-d2/d4-d5/a5,-(sp)
        clr.l   d0
        clr.l   d1
;
        bsr     smuladd   ;obtain sum of products
        moveq.l #1,d6
        clr.l   d4
        clr.l   d5
dc_power1
        moveq.l #-1,d2
        move.w  (a5),d3
        muls    -2(a5),d3
        bmi.s   dc_power2
        clr.l   d2
dc_power2:
        add.l   d3,d5
        addx.l  d2,d4
        movea.l a1,a5
        subq.l  #6,a5             ;Y0(chan2/chan4)
        dbra    d6,dc_power1
        asr.l   #1,d4     ;divide by 2
        roxr.l  #1,d5
        add.l   d5,d1
        addx.l  d4,d0
        move.b  #' ',d3
        move.b  #'-',d3           ;negative sign
        neg.l   d1                ;compute 2's complement
        negx.l  d0
        clr.l   d2
        moveq.l #9,d4
dc_power4:
        asr.l   #1,d0      ;divide result by 1024
        roxr.l  #1,d1      ;quotient in d1
        roxr.l  #1,d2
        dbra    d4,dc_power4
    tst.l     d2
    bpl.s     dc_power5
    addq.l    #1,d1      ;round up
dc_power5:
        lsr.l   #6,d2   ;remainder in upper word of d2
        move.l  d1,d0   ;since result is less than 32 bits
;
    movem.l   (sp)+,d1-d2/d4-d5/a5
;
    rts
;
```

```
;************************************
; Subroutine phase3
; purpose: to compute power and display
;       power for phases 1 and 2
; initial
; conditions:
; registers
; affected:  don't care
;************************************
;
phase3:
    addq.l    #8,a1          ;point to Y1 chan 1
    movea.l   a1,a4
    movea.l   a5,a2
    movea.l   a5,a3
    addq.l    #2,a4          ;point to Y1 chan 2
    subq.l    #6,a2          ;point to Yn chan 1
    subq.l    #4,a3          ;point to Yn chan 2
    bsr  correct
    bsr  evaluate
    move.l    4(a0),d0
    move.l    d0,power1
    bsr  watt_bcd
    bclr.b    #1,state
    move.l    a5,-(sp)  ;prepare to display
        lea.l   disp_buff1,a5
    move.l    #'W1= ',(a5)+
       bsr     update
    bsr  blank
    move.b    (a6),2(a6)     ;move termination character
    move.b    #'W',1(a6)
    move.b    -1(a6),(a6)
    move.b    #'.',-1(a6)    ;insert decimal point
    move.l    (sp)+,a5
    addq.l    #4,a1          ;point to Y1 chan 3
    addq.l    #4,a4          ;point to Yn chan 4
    addq.l    #4,a2          ;point to Y1 chan 3
    addq.l    #4,a3          ;point to Yn chan 4
;
    bsr  correct
    bsr  evaluate
    move.l    4(a0),d0
    move.l    d0,power2
    bsr  watt_bcd
    bclr.b    #2,state
    move.l    a5,-(sp)  ;prepare to display
        lea.l   disp_buff2,a5
    move.l    #'W2= ',(a5)+
       bsr     update
    bsr  blank
    move.b    (a6),2(a6)     ;move termination character
    move.b    #'W',1(a6)
```

```
    move.b    -1(a6),(a6)
    move.b    #'.',-1(a6)    ;insert decimal point
    move.l    (sp)+,a5
    rts
;
;**********************************
;   Subroutine  pf_100
; purpose : to compute 100 * pf (power factor)
; registers
; affected: none
; final
; conditions: bcd value of 100 *pf in pf_store
;**********************************
;
pf_100:
    movem.l   d0-d5,-(sp)
    move.l    Vmsqr,d0
    move.l    Imsqr,d1
    bsr  mul32_32  ;sqr(V)*sqr(I)
    bsr  pflt      ;convert 64_bit result to float
    move.l    d0,d4
    move.l    d1,d5
    move.l    Watt,d0
    move.l    d0,d1
    bsr  mul32_32  ;sqr(W)
    bsr  pflt      ;convert to float
    move.l    #$40c38800,d2
    clr.l     d3        ;into d2-d3
    bsr  pmul
    move.l    d4,d2
    move.l    d5,d3
    bsr  pdiv
    bsr  pfix      ;convert to fixed point
    move.l    d1,d0          ;result=[0,10000]
    bsr  sqrt
    move.w    d0,d1          ;result=[0,100]
    clr.l     d0
    bsr  bcd1      ;bcd value in lower word of d0
    move.w    d0,pf_store
    movem.l   (sp)+,d0-d5
    rts
;
;**********************************
;   Subroutine  pf3_100
; purpose : to compute 100 * pf (power factor)
;       for 3-phase case
; formula= sqrt(1exp4*sqr(P1+P2)/(sqr(P1+P2)+3*sqr(P2-P1))
; registers
; affected: none
; final
; conditions: bcd value of 100 *pf in pf_store
;**********************************
```

```
pf3_100:
    movem.l   d0-d5,-(sp)
    move.l    power1,d0
    move.l    power2,d2
    sub.l     power2,d0
    bne.s   pf3_1001
    moveq.l   #100,d1          ;trivial case
    bra  pf3_1003
pf3_1001:
    bpl.s     pf3_1002
    neg.l     d0          ;compute 2's complement
pf3_1002:               ;[P2-P1]
    add.l     power2,d2
    move.l    d0,d1           ;[P2-P1]
    asl.l     #1,d0           ;2*[P2-P1]
    add.l     d1,d0           ;3*[P2-P1]
    bsr  mul32_32  ;3*[P2-P1]*[P2-P1]
    move.l    d0,d4           ;save result
    move.l    d1,d5
    move.l    d2,d0           ;P2+P1
    move.l    d2,d1           ;P2+P1
    bsr  mul32_32
    add.l     d1,d5
    addx.l    d0,d4
    bsr  pflt       ;convert to float
    move.l    #$40c38800,d2
    clr.l     d3          ;into d2-d3
    bsr  pmul       ;1exp4*sqr(P2+P1)
    exg.l     d0,d4
    exg.l     d1,d5
    bsr  pflt       ;convert to float
    move.l    d0,d2
    move.l    d1,d3
    move.l    d4,d0
    move.l    d5,d1
    bsr  pdiv
    bsr  pfix
    move.l    d1,d0
    bsr  sqrt       ;compute 100*pf
    move.w    d0,d1
pf3_1003:
    clr.l     d0
    bsr  bcd1       ;bcd value in lower word of d0
    move.w    d0,pf_store
    movem.l   (sp)+,d0-d5
    rts
;
;*********************************
; subroutine watt_bcd
; purpose: to compute bcd value of power
; initial     d0 contains number
; conditions: d3 holds sign
```

```
; registers
; affected:   d0
;*********************************
;
watt_bcd:
        movem.l  d1/d2,-(sp)        ;save d1/d2
    clr.l     d1
    clr.l     d2
    btst.b    #1,state
    beq.s     watt_bcd1
    move.w    denom_w1,d2
    move.w    numer_w1,d1
    bsr  mul32_32
    bra.s     watt_bcd3
watt_bcd1:
    btst.b    #2,state
    bne.s     watt_bcd2
    movea.l   #7,a2           ;TEST
    bra  ramerr
watt_bcd2:
    move.w    denom_w2,d2
    move.w    numer_w2,d1
    bsr  mul32_32
watt_bcd3:
    tst.w     d2
    beq.s     watt_bcd4
    subq.w    #1,d2           ;d2 < 16 bits
watt_bcd3a:
    asr.l     #1,d0
    roxr.l    #1,d1
    dbra d2,watt_bcd3a
    bcc.s     watt_bcd3b
    addq.l    #1,d1           ;round up
watt_bcd3b:
    clr.l     d0        ;prepare for routine
    bsr  bcd1      ;convert to bcd
watt_bcd4:
        movem.l  (sp)+,d1/d2    ;restore d1/d2
;
    rts
;
;
;*********************************
; subroutine   sc_fact
; purpose : to  enter new scale factors
;*********************************
;
sc_fact:
        movem.l d0-d7/a0-a6,-(sp)
   bset.b    #4,state1 ;set scale factor change semaphore
   bsr  clear
   lea  key_buff,a0
```

```
        moveq     #line3,d3
        lea  messg27(pc),a5 ;display ' TNUM V1 HIGH'
        bsr  display
            bsr     acq_disp
        lea  tnumer_v1h,a1
            bsr     acq
        move.w    (a1),tnumer_v1h
        bsr  clear
        lea  key_buff,a0
        moveq     #line3,d3
        lea  messg28(pc),a5 ;display ' TNUM V1 LOW'
        bsr  display
            bsr     acq_disp
        lea  tnumer_v1l,a1
            bsr     acq
        move.w    (a1),tnumer_v1l
        bsr  clear
        lea  key_buff,a0
        moveq     #line3,d3
        lea  messg29(pc),a5 ;display ' TNUM V2 HIGH'
        bsr  display
            bsr     acq_disp
        lea  tnumer_v2h,a1
            bsr     acq
        move.w    (a1),tnumer_v2h
;
        bsr  clear
        lea  key_buff,a0
        moveq     #line3,d3
        lea  messg30(pc),a5 ;display ' TNUM V2 LOW'
        bsr  display
            bsr     acq_disp
        lea  tnumer_v2l,a1
            bsr     acq
        move.w    (a1),tnumer_v2l
;
        bsr  clear
        lea  key_buff,a0
        moveq     #line3,d3
        lea  messg31(pc),a5 ;display ' TNUM I1'
        bsr  display
            bsr     acq_disp
        lea  tnumer_i1,a1
            bsr     acq
        move.w    (a1),tnumer_i1
;
        bsr  clear
        lea  key_buff,a0
        moveq     #line3,d3
        lea  messg32(pc),a5 ;display ' TNUM I2'
        bsr  display
            bsr     acq_disp
```

```
    lea  tnumer_i2,a1
         bsr     acq
    move.w    (a1),tnumer_i2
;
    bsr  watt_fact
    bsr  store_fact     ;update voltage,current and power
;                  scale factor numerators
    bclr.b    #4,state1 ;clear    scale    factor    change
semaphore
;
    lea  key_buff,a0
    move.b    #$ff,(a0)
;
         movem.l (sp)+,d0-d7/a0-a6
    rts
;
;
;*********************
; subroutine shunt
; purpose: to update the shunt values
;**********************
;
shunt:
         movem.l d0-d7/a0-a6,-(sp)
shunt_disp:
    lea  key_buff,a0
    move.b    #$ff,(a0)
    bsr  clear
         moveq    #line1,d3
         lea      messg48(pc),a5    ;enter shunts
         bsr      display
         moveq    #line2,d3
         lea      messg48a(pc),a5   ;1 for channel 2
        bsr     display
         moveq    #line3,d3
         lea      messg48b(pc),a5   ;2 for channel 4
         bsr      display
         moveq    #line4,d3
         lea      messg48c(pc),a5   ;0 to skip
         bsr      display
    bset.b    #5,state1
    lea  key_buff,a0
;
    tst.b   prog_state
         beq.s   shunt0
;
    bsr  program        ;in progrm mode
;
shunt0:
    tst.b     (a0)
    beq  shunt3
    cmpi.b    #1,(a0)
```

```
    beq.s     shunt1
    cmpi.b    #2,(a0)
    bne.s     shunt0
;
; channel 4 shunt value
;
    move.b    #$ff,(a0)
    bsr  clear
    moveq     #line3,d3
    lea  messg26(pc),a5
    bsr  display
        bsr     acq_disp
    lea  shunt_val2,a1
        bsr     acq
    move.w    (a1),shunt_val2
    movem.l   d0/d1,-(sp)
    clr.l     d0
    move.w    tnumer_i2,d0
    mulu.w    (a1),d0
    move.l    #100,d1          ;initialized  with  a  10amp
shunt
    divu d1,d0
    tst.l     d0
    bpl.s     shunt0a
    addq.w    #1,d0            ;correct to nearest integer
shunt0a:
    move.w    d0,tnumer_i2
;
    movem.l   (sp)+,d0/d1
    bra  shunt_disp
;
; channel 2 shunt value
;
shunt1:
    move.b    #$ff,(a0)
    bsr  clear
    moveq     #line3,d3
    lea  messg26(pc),a5
    bsr  display
        bsr     acq_disp
    lea  shunt_val1,a1
        bsr     acq
    move.w    (a1),shunt_val1
    movem.l   d0/d1,-(sp)
    clr.l     d0
    move.w    tnumer_i1,d0
    mulu.w    (a1),d0
    move.l    #100,d1          ;initialized  with  a  10amp
shunt
    divu d1,d0
    tst.l     d0
    bpl.s     shunt1a
```

```
    addq.w    #1,d0          ;correct to nearest integer
shunt1a:
    move.w    d0,tnumer_i1
;
    movem.l   (sp)+,d0/d1
    bra  shunt_disp
;
shunt3:
    bsr.s     watt_fact
    bsr  store_fact     ;update voltage,current and power
;                   scale factor numerators
    move.b    #$ff,(a0)
    bclr.b    #5,state1 ;disable shunt entry
;                   mode semaphore
        movem.l (sp)+,d0-d7/a0-a6
    rts
;
;*******************************
; subroutine watt_fact
; purpose : to compute numerators of power
;        scale factors
; registers   none
; affected:
;*******************************
;
watt_fact:
    movem.l   a1-a3,-(sp)
    lea  tnumer_i1,a1
    lea  tnumer_v1h,a2
    lea  tnumer_w1h,a3
    bsr  init_fact ;compute tnumer_w1h
    lea  tnumer_i1,a1
    lea  tnumer_v1l,a2
    lea  tnumer_w1l,a3
    bsr  init_fact ;compute tnumer_w1l
    lea  tnumer_i2,a1
    lea  tnumer_v2h,a2
    lea  tnumer_w2h,a3
    bsr  init_fact ;compute tnumer_w2h
    lea  tnumer_i2,a1
    lea  tnumer_v2l,a2
    lea  tnumer_w2l,a3
    bsr  init_fact ;compute tnumer_w2l
    movem.l   (sp)+,a1-a3
    rts
;
; floating point routines
;
;   pfix: fix a floating point argument
;   pflt: float a fixed point argument
;   pmul: multiply 2 floating point numbers
;   pdiv: divide 2 floating point number
```

```
include pfix
include pflt
include pmul
include pdiv
;
;
;***************************
;
; interrupt service routines
;
;***************************
;
;
int_key:
        movem.l d0-d7/a0-a6,-(a7)
;
        move.b  keyb+2,d0    ;read 8279 status
        move.b  keyb,key_buff  ;read and store
;                                key code
        movem.l (a7)+,d0-d7/a0-a6
        rte
;
;*****************
int_tim:
        movem.l d0-d7/a0-a6,-(sp)
        lea     time,a5
        btst.b  #0,status(a5)
        beq     intp2e
;
        move.b  #%10100011,ctl2(a5)  ;point to timer 1
;
        move.b  #%10000011,ctl13(a5)  ;reset  timer
        clr.b   ac_mode
;
intp2e  movep.w cnt2h(a5),d0
        movep.w cnt3h(a5),d0
        movem.l (a7)+,d0-d7/a0-a6
        rte
;
;****************
;
int_gpia:
        movem.l d0-d7/a0-a6,-(sp)
        lea.l   gpia,a0
        move.b  instatr(a0),d0
        btst.l  #7,d0
        beq     return
;
        move.w  d0,ccr
        bcs.s   bi_0              ;DATA available?
        bra     return
;
```

```
bi_0    andi.b  #%11111100,inmskr(a0)
        move.l   prog_max,a1
;
bi_1    move.b  instatr(a0),d0
        move.w  d0,ccr
        bcc.s   bi_1
        bvs.s   end_1
        move.b  dtinr(a0),d0
;
        lea.l   prog_table(pc),a2
        lea.l   key_table-1(pc),a3
;
    cmpi.b    #'T',d0         ;^D to end
    beq.s     bi_2      ;if end code not sent, display
;                   error message and receive new byte
    cmpi.b    #'U',d0         ;^E to undo
    subq.l    #1,a3
    bra.s     bi_1      ;receive new byte
decode_1:
        cmp.b   (a2)+,d0
        bne.s   decode_0
    move.l    a1,prog_max
        bra.s   bi_1
;
bi_2     bsr  clear
    movem.l   d3/a5,-(sp)
    bsr  display
    movem.l   (sp)+,d3/a5
    bra.s     bi_1      ;receive new byte
;
rem_0   ori.b   #local,pia_b
    bclr.b    #7,pia_b  ;turn off remote led (no 8)
        clr.b   prog_state      ;indicate device in normal
mode
        move.b  pia_b,piadb
        bra.s   return
;
spas_1  btst.b   #2,cstatr(a0)    ;wait for controller to
send the SPD
        beq.s   atn             ;command thereby releasing
the GPIA
;                                  from the SPAS
        move.b  spolr(a0),spolr(a0)      ;Clear the "rsv"
bit
;                                          fall through to
"return"
;
return  move.b  #0,instatr(a0)
        move.b  #%11111111,inmskr(a0)
        movem.l (a7)+,d0-d7/a0-a6
;
        rte
```

```
;
prog_table:
        dc.b    '0123'
        dc.b    '4567'
        dc.b    '89AB'
        dc.b    'CDEF'
; control codes
        dc.b    'GHIJ'
        dc.b    'KLMN'
        dc.b    'OPQR'
        dc.b    'STUV'
;
key_table:
        dc.b    0,1,2,3
        dc.b    8,9,$a,$b
        dc.b    $10,$11,$12,$13
        dc.b    $18,$19,$1a,$1b
; control codes
        dc.b    $80,$81,$82,$83
        dc.b    $88,$89,$8a,$8b
        dc.b    $90,$91,$92,$93
        dc.b    $98,$99,$9a,$9b
;
;****************
;
;  other exceptions
;
b_excpt bsr.s   excp1
a_excpt bsr.s   excp1
i_excpt bsr.s   excp2
d_excpt bsr.s   excp2
c_excpt bsr.s   excp2
o_excpt bsr.s   excp2
p_excpt bsr.s   excp2
t_excpt bsr.s   excp2
x_excpt bsr.s   excp2
y_excpt bsr.s   excp2
s_excpt bsr.s   excp2
;
int     bsr.s   excp2
int7    bsr.s   excp2
;
excp1   movem.l d0/a0,-(sp)
        move.l  22(sp),a0
        move.w  18(sp),d0
        cmp.w   -(a0),d0
        beq.s   excp10
        cmp.w   -(a0),d0
        beq.s   excp10
        cmp.w   -(a0),d0
        beq.s   excp10
        cmp.w   -(a0),d0
```

```
        beq.s   excp10
        subq.l  #2,a0
excp10  move.l  a0,22(sp)
        movem.l (sp)+,d0/a0
        move.l  (sp),8(sp)
        addq.l  #8,sp
excp2   move.l  6(sp),disp_buff
        move.l  (sp)+,a1
;
        lea.l   b_excpt(pc),a5
        suba.l  a5,a1
        lea.l   abotab(pc),a2
        move.l  a1,d0
        subq.l  #2,d0
        move.w  0(a2,d0.l),a5
        adda.l  a2,a5
        bsr     clear
        moveq   #line1,d3
        bsr     display
;
        lea.l   disp_buff,a5
        moveq.l #4,d5
        bsr     hex_ascii       ;convert to ascii
;
        moveq   #line3,d3
        lea.l   messg11(pc),a5
        bsr     display
;
        moveq   #line4,d3
        lea.l   disp_buff,a5
        bsr     display
;
excp3   bra.s   excp3
        rte
;
;
;***************
;  messages
;*****************
;
messg1  dc.b    '  MEMORY TEST',0
messg2  dc.b    '  RAM TEST FAILURE ',0
messg3  dc.b    'A: RECAL  B: SAMPLE',0
messg4  dc.b    'C: CALIBRATE',0
messg5  dc.b    'D: ACQ. SIZE',0
messg6  dc.b    'E: DELAY   F: FREQ.',0
messg7  dc.b    '  ADDRESS:',0
messg8  dc.b    '  FREQUENCY IN Hz',0
messg8a dc.b    '              ACTUAL',0
messg8b dc.b    '              MIN',0
messg8c dc.b    '              MAX',0
messg9  dc.b    '  CALIBRATE AC ',0
```

```
messg9a dc.b    ' CALIBRATE RMS ',0
messg10 dc.b    '  CALIBRATE DC ',0
messg10a dc.b   '        DC        ',0
messg11 dc.b    '  CORRECT PC VALUE:',0
messg12 dc.b    '    LOW',0
messg13 dc.b    '  1 FOR CHANNEL 1',0
messg14 dc.b    '  2 FOR CHANNEL 2',0
messg15 dc.b    '  3 FOR CHANNEL 3',0
messg16 dc.b    '  4 FOR CHANNEL 4',0
messg17 dc.b    '    PRESS:',0
messg18 dc.b    '  1 FOR PHASE 1',0
messg19 dc.b    '  2 FOR PHASE 2',0
messg19a dc.b   '  3 FOR 3-PHASE',0
messg20 dc.b    '       UUCG',0
messg21 dc.b    '       UACG',0
messg22 dc.b    ' ENTER 4 DIGIT NO.',0
messg23 dc.b    '0000 = CONT. CASE',0
messg23a dc.b 'SHUNT VALUE * 10',0
messg24 dc.b    '  SAMPLE SIZE',0
messg25 dc.b    '      DELAY',0
messg26  dc.b '                    ',0
messg27 dc.b    '  TNUM V1 HIGH',0
messg28 dc.b    '  TNUM V1 LOW',0
messg29 dc.b    '  TNUM V2 HIGH',0
messg30 dc.b    '  TNUM V2 LOW',0
messg31 dc.b    '     NUM I1',0
messg32 dc.b    '     NUM I2',0
;
messg35  dc.b ' OVERLOAD',0
messg36  dc.b '0:   RESET',0
messg37  dc.b '1: DISPLAY ON/OFF',0
messg38  dc.b '^F: NEXT MENU',0
messg39  dc.b '^B: PREVIOUS MENU',0
messg40  dc.b '2:CHAN 1 HIGH/LOW',0
messg41  dc.b '3:CHAN 3 HIGH/LOW',0
messg42  dc.b '4:O_LOAD TEST ON/OFF',0
messg43  dc.b '5: NUMERATOR (S/F)',0
messg43a dc.b 'CHANGE SC/FACTORS',0
messg44  dc.b '6:DENOMINATOR (S/F)',0
messg45  dc.b '7:TRIG SRC CHAN1/2',0
messg46  dc.b '8: CLEAR DATA STORE',0
messg46a dc.b ' LINES REMAINING',0
messg47  dc.b '9: CALIBRATE RMS',0
messg48  dc.b '^0:ENTER SHUNT VALS',0
messg48a dc.b   '  1 FOR CHANNEL 2',0
messg48b dc.b   '  2 FOR CHANNEL 4',0
messg48c dc.b   '  0 TO EXIT',0
messg48d dc.b   '  1 TO CONTINUE',0
messg49  dc.b '^1:LINES IN RAM',0
messg50  dc.b '^2:DISPLAY RAM DATA',0
messg51  dc.b '^3:STORAGE ON/OFF',0
messg52  dc.b '^4:KEYPAD PROGRAM',0
```

```
messg53  dc.b '^5:CORRECT ENTRY',0
messg54  dc.b '^6:END ENTRY',0
messg55  dc.b '^7:CHANGE SC/FACTS',0
messg56  dc.b '^8:RECAL ALL CHANS',0
messg57  dc.b '^9:',0
messg58  dc.b '^A:',0
messg59  dc.b '^D:END PROG ^E:UNDO',0
messg60  dc.b ' METER READY',0
messg61  dc.b '^C: RETURN TO',0
messg62  dc.b '      MENU',0
messg63  dc.b 'GPIB ERROR 1',0
;
ab1      dc.b    '  bus error',0
ab2      dc.b    ' Address error',0
ab3      dc.b    'illegal instruction',0
ab4      dc.b    'division by zero',0
ab5      dc.b    'chk exception',0
ab6      dc.b    'trapv exception',0
ab7      dc.b    'privilege violation',0
ab8      dc.b    'trace....',0
ab9      dc.b    'ill instruct (1011)',0
ab10     dc.b    'ill instruct (1111)',0
ab11     dc.b    'spurious interrupt',0
ab12     dc.b    'unexpected interrupt',0
ab13     dc.b    'level 7 interrupt',0
;
abotab   dc.w    (ab1-abotab)
         dc.w    (ab2-abotab)
         dc.w    (ab3-abotab)
         dc.w    (ab4-abotab)
         dc.w    (ab5-abotab)
         dc.w    (ab6-abotab)
         dc.w    (ab7-abotab)
         dc.w    (ab8-abotab)
         dc.w    (ab9-abotab)
         dc.w    (ab10-abotab)
         dc.w    (ab11-abotab)
         dc.w    (ab12-abotab)
         dc.w    (ab13-abotab)
;
;
;
         section  1
;
smul_store       ds.l    2
samp_end         ds.l    1
rms_temp ds.l 1     ;holds correction for rms
disp_buff        ds.b    24
disp_buff1       ds.b    24
disp_buff2       ds.b    24
disp_buff3       ds.b    24
key_buff         ds.b    1          ;key buffer
```

```
pia_a           ds.b    1     ;state of port a
pia_b           ds.b    1      ;state of port b
state           ds.b    1
state1          ds.b    1
;
_flterr         ds.b    1
auto_chan     ds.l 1
cal_buff1       ds.w    1     ;calibrated value (channel 1)
cal_buff2       ds.w    1     ;calibrated value (channel 2)
cal_buff3       ds.w    1     ;calibrated value (channel 3)
cal_buff4       ds.w    1     ;calibrated value (channel 4)
cal_chan        ds.b    1     ;channel to calibrate
freq_state      ds.b    1     ;bit 0= min flag  bit 1= max
flag
freq_min        ds.b    2     ;minimum freq  count value
freq_max        ds.b    2     ;maximum freq  count value
alpha16_store   ds.b    2     ;16*delta  store
ac_mode         ds.b    1
prog_state      ds.b    1     ;$ff  =>  read  program  from
prog_buff
prog_buff       ds.b    128  ;buffer for program bytes
prog_pointer  ds.l    1 ;points  to  next  program  byte
required
prog_max ds.l    1 ;points  to  next  free  location  in
prog_buff
acq_count       ds.l    1       ;number of acquisitions
delay_value     ds.l    1       ;delay value
delay_count      ds.l     1         ;present delay counter
state
;
numer_v1 ds.l 1     ;numerator for V1 scale factor
numer_i1 ds.l 1     ;numerator for I1 scale factor
numer_v2 ds.l 1     ;numerator for V2 scale factor
numer_i2 ds.l 1     ;numerator for I2 scale factor
numer_w1 ds.l 1     ;numerator for W1 scale factor
numer_w2 ds.l 1     ;numerator for W2 scale factor
denom_v1 ds.l 1     ;denominator for V1 scale factor
denom_i1 ds.l 1     ;denominator for I1 scale factor
denom_v2 ds.l 1     ;denominator for V2 scale factor
denom_i2 ds.l 1     ;denominator for I2 scale factor
denom_w1 ds.l 1     ;denominator for W1 scale factor
denom_w2 ds.l 1     ;denominator for W2 scale factor
;
;
tnumer_v1h    ds.l 1    ;temp numerator for V1 high scale
factor
tnumer_v1l    ds.l 1    ;temp numerator for V1 low scale
factor
tnumer_i1     ds.l 1    ;temp numerator for I1 scale factor
tnumer_v2h    ds.l 1    ;temp numerator for V2 high scale
factor
tnumer_v2l    ds.l 1    ;temp numerator for V2 scale factor
```

```
tnumer_i2       ds.l 1      ;temp numerator for I2 scale factor
tnumer_w1h      ds.l 1      ;temp numerator for W1 high scale
factor
tnumer_w1l      ds.l 1      ;temp numerator for W1 low scale
factor
tnumer_w2h      ds.l 1      ;temp numerator for W2 high scale
factor
tnumer_w2l      ds.l 1      ;temp numerator for W2 low scale
factor
;
;
shunt_val1      ds.l 1      ;holds 10 * shunt value (chan 2)
shunt_val2      ds.l 1      ;holds 10 * shunt value (chan 4)
Vmax            ds.w 1      ;holds max Voltage sample no.
Imax            ds.w 1      ;holds max Current sample no.
Vmsqr           ds.l 1      ;holds mean square voltage value
Imsqr           ds.l 1      ;holds mean square current value
Watt            ds.l 1      ;holds power value
pf_store ds.w 1     ;holds 100*pf
power1          ds.l 1      ;holds phase 1 power in 3-phase
mode
power2          ds.l 1      ;holds phase 2 power in 3-phase
mode
recal_point     ds.w 1      ;point to next free location
;                       in recal_buff
recal_buff      ds.w 256    ;storage for 256 values
menu_state      ds.b 1      ;current menu pointer
disp_state      ds.b 1      ;display state
meas_state      ds.b 1      ;measurement state
meas_state1     ds.b 1      ;measurement state1
;                       1 => display 3 lines
store_size      ds.w 1      ;holds number of lines to be
displayed
data_pointer    ds.l 1      ;points to current storage location
;
        end
;
```